1995

Diagnostic and adaptive redundant robotic planning and control

Anya Lynn Tascillo
Binghamton University--SUNY

DIAGNOSTIC AND ADAPTIVE REDUNDANT ROBOTIC PLANNING AND CONTROL

BY

ANYA LYNN TASCILLO

B.S.M.E., Rensselaer Polytechnic Institute
Troy, NY 1988

M.S.M.S., Rensselaer Polytechnic Institute
Troy, NY 1992

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate School of the
State University of New York
at Binghamton
1995

Accepted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate School of the
State University of New York
at Binghamton

Nikolaos G. Bourbakis [illegible] April 17, 1995
Associate Professor and Advisor
Department of Electrical Engineering

George J.Klir George Klir April 17, 1995
Distinguished Professor
Department of System Science and Industrial Engineering

N. Eva Wu [illegible] April 17, 1995
Associate Professor
Department of Electrical Engineering

ABSTRACT

Neural networks and fuzzy logic are combined into a hierarchical structure capable of planning, diagnosis, and control for a redundant, nonlinear robotic system in a real world scenario. Throughout this work levels of this overall approach are demonstrated for a redundant robot and hand combination as it is commanded to approach, grasp, and successfully manipulate objects for a wheelchair-bound user in a crowded, unpredictable environment. Four levels of hierarchy are developed and demonstrated, from the lowest level upward: diagnostic individual motor control, optimal redundant joint allocation for trajectory planning, grasp planning with tip and slip control, and high level task planning for multiple arms and manipulated objects. Given the expectations of the user and of the constantly changing nature of processes, the robot hierarchy learns from its experiences in order to more efficiently execute the next related task, and allocate this knowledge to the appropriate levels of planning and control. The above approaches are then extended to automotive and space applications.

ACKNOWLEDGEMENTS

My eternal gratitude to:

Professor N.Bourbakis for his support and encouragement to complete my thesis,

Professors G.Klir and E.Wu for their patience and insights,

Mark and Mom for being there while I beat the neural nets into submission,

Joe, Ilene, Jiri, and Monika for squeezing glimpses of a social life in amongst the chaos,

Ranga, Mo, Vince, Joe, Alex, Ron, Kashif, Kevin, Jim, Steve, and the rest of the lab rats for all of the memories late at night and under the wire,

Kevin, Antonio, and the rest of the campus security crew for keeping us safe, secure, and entertained during those 16 hour sessions in the lab,

Csilla and Jim for capably passing the driving robot torch (and answering my continual questions afterward),

Earl for his endurance, support, and advice as friend and human driver yardstick,

The many Ford APTL (and Dyno) operators, technicians, engineers, supervisors, and staff for enduring my robots and me (especially those who endured my rough draft),

The many requestors who loaned me vehicles, not knowing what they might get back, and finally

My cat Dusty, for killing the bat on Mark's papers instead of mine.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Conventional feedback control[1] attempts to maintain the stable, accurate performance of an assumed time invariant system despite external disturbances. Optimal control[2] selects the feedback parameters, or gains, by optimizing more than one aspect of output performance, such as both time and energy expenditure, from initial to final conditions. Robust control[3,4] chooses feedback gains that will ensure stable operation as parameters within the system, or controlled plant, vary. When robust control cannot offer high enough output accuracy across the anticipated range of plant variability, adaptive control[5] modifies the controller parameters themselves via a performance index. Model reference control[6] is a special form of adaptive control whose controller feedback mechanism attempts to steer the performance of a plant toward that of a desired model.

Traditionally, these methods have required a sufficiently accurate linear model of the plant, and all unknown or insufficiently modelled parameters are either approximated statistically or treated as disturbances. These approaches have still been inadequate for high speed, high accuracy applications where the controlled plant nonlinearly drifts with time. Although approaches such as the Extended Kalman Filter attempt to make the next estimation of a parameter conform to a nonlinear trajectory, such as along an ellipse, an assumption such as this cannot always be made.[7]

As customers demand greater reliability and consistency in product operation, control procedures must evolve to accommodate the time dependent, noisy nonlinearities that were tolerated, ignored, or roughly approximated in the past. Fuzzy logic and neural networks can help the engineer understand the interdependencies of nonlinear behavior, by directly modelling the input/output relationship, diagnosing a variable responsible for parameter drift, or allocating resources to more effectively plan for uncertainty.

This work will explore the role fuzzy logic and neural networks, in conjunction with expert systems, can play when a redundant manipulator hand (two fingers, one thumb, two independent actuators per appendage) and robot combination with DC joint motors is commanded to approach, grasp, and successfully manipulate an object. The goal is to be able to integrate a functionally complete planning and control strategy into an algorithm that can run in real time for interaction with humans. As a frame of reference for assumptions and an illustrative example, the redundant robot and hand combination are assumed to be mounted to the arm of a wheelchair. As an aid to the disabled, the robot will be commanded to perform tasks such as lifting a pencil or retrieving a soda can from a cluttered refrigerator.

Although visual feedback will contribute vital information for any scenario, the primary source of diagnosis and planning for the manipulation subtask will be tactile feedback through the hand's fingers. The touch feedback will be considered crucial from the standpoint not only of tip and slip control (where fuzzy techniques lead to neural), but will continue to influence the grasp approach vector itself via the training of object grasp category networks, which rank the earlier attempts made to grasp a class of objects and learn which to try next and save time in the future.

In any complex scenario, there is no one performance measure that will assure successful completion of a task. To this end, control and planning are arranged into a four level hierarchy, each sharing information with other levels, but focussed on a specific combination of a performance criterion. Given the expectations of the user and of the constantly changing nature of processes, the robot hierarchy learns from its experiences in order to more efficiently execute the next related task, and allocate this knowledge to the appropriate levels of planning and control.

Various adaptive learning and control approaches are reviewed, and neural network training and architectures are compared in Chapter 1. Some of these approaches are investigated and compared in Chapter 2, where the neural network architecture developed is included as part of a larger planning and control hierarchy with expert systems and fuzzy logic. Each level of this new hierarchy is developed and demonstrated in subsequent chapters, from local to abstract: diagnostic motor control in Chapter 3, optimal redundant joint allocation for trajectory planning in Chapter 4, grasp planning with tip and slip control in Chapter 5, and high level task planning for multiple arms and manipulated objects in Chapter 6. The above approaches are then extended to an automotive application in Chapter 7 to demonstrate extensibility of the approach.

References

[1]Kuo, Benjamin C., **Automatic Control Systems**. Englewood Cliffs, N.J.: Prentice Hall, 1991.

[2]Lewis, Frank L., **Optimal Control**. New York: John Wiley and Sons, 1986.

[3]Doyle, John C., Bruce A. Francis, and Allen R. Tannenbaum, **Feedback Control Theory**. New York: Macmillan Publishing Company, 1992.

[4]Francis, Bruce A., **A Course in H_∞ Control Theory.** New York: Springer-Verlag, 1987.

[5]Astrom, K.J. and B. Wittenmark, **Adaptive Control**. New York: Addison-Wesley Publishing Company, 1989.

[6]Landau, Yoan D., **Adaptive Control: The Model Reference Approach**. New York: Marcel Dekker, Inc., 1979.

[7]Gelb, Arthur, ed., **Applied Optimal Estimation**. Cambridge, Massachusetts: The M.I.T. Press, 1989.

CHAPTER 1:

ADAPTIVE LEARNING AND CONTROL APPROACHES

1.1 The Adaptive Learning and Control Problem

When the redundant robot of Figure 1 is given this complex control scenario, there is no guarantee that any set of commands will be repeated, and if they are, there is no guarantee that the system will be operating under the same internal parameters and external environmental conditions. The implemented planning mechanism chosen must be flexible to instantaneous changes of goal state, and the overall adaptive control mechanism chosen must provide a response that is stable and within acceptable limits for any given reference command, despite parameter drift and both anticipated and unanticipated disturbances. As the user of this system is likely to lose patience waiting for it to adapt to a new situation, the robot must incorporate learning into its planning and control.

In many situations, classical and modern control theory is inadequate to control complex systems. The system to be controlled may be partially or totally unknown, parameters within the plant that is modelled may drift, and highly nonlinear behavior within the system may drive an algorithm based upon linearization unstable. A primary

CHAPTER 1:
ADAPTIVE LEARNING AND CONTROL APPROACHES

1.1 The Adaptive Learning and Control Problem

When the redundant robot of Figure 1 is given this complex control scenario, there is no guarantee that any set of commands will be repeated, and if they are, there is no guarantee that the system will be operating under the same internal parameters and external environmental conditions. The implemented planning mechanism chosen must be flexible to instantaneous changes of goal state, and the overall adaptive control mechanism chosen must provide a response that is stable and within acceptable limits for any given reference command, despite parameter drift and both anticipated and unanticipated disturbances. As the user of this system is likely to lose patience waiting for it to adapt to a new situation, the robot must incorporate learning into its planning and control.

In many situations, classical and modern control theory is inadequate to control complex systems. The system to be controlled may be partially or totally unknown, parameters within the plant that is modelled may drift, and highly nonlinear behavior within the system may drive an algorithm based upon linearization unstable. A primary

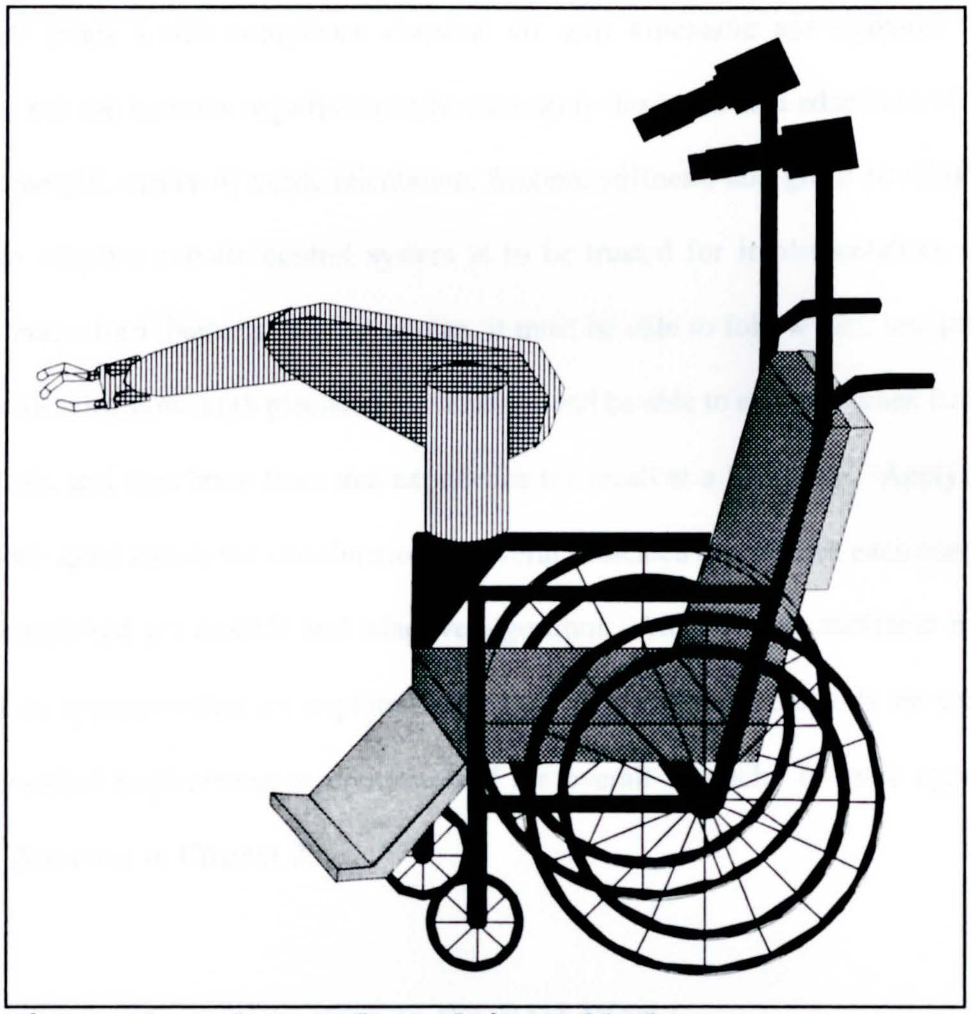


Figure 1: Robot and Hand installed on a wheelchair.

example of such a complex system is a robotic hand. Not only are robotic hands redundant joints which complicate classical six axis kinematic and dynamic control schemes, but the motions required must be intricately dexterous and adaptive to loads of variable weight, center of mass, orientation, friction, stiffness, and grasp stability.

If an adaptive robotic control system is to be trusted for implementation, it must reliably outperform humans. Like a human, it must be able to follow fast, low precision movements with slow, high precision movements, and be able to estimate when first faced with a task, and then learn from that experience for recall at a later time. Applying this to a robotic hand entails the coordination of several dedicated algorithms, each controlling with a simplified yet flexible and adaptive algorithm. After several nonlinear methods of function approximation are explored and neural network based methods are expanded upon, practical implementation concerns and the overall hierarchy of these approaches will be discussed in Chapter 2.

1.2 Approximating a Function in the Real World

Whether the problem concerns determination of acceptable control parameters, stable grasps, optimal configurations, or timely process flows, all can be simplified to the task of learning a functional relationship. Each nonlinear approach considered for this task will be discussed as to its utility in a real time, cluttered, and noisy environment.

1.2.1 Polynomial Approximations

A popular first attempt at nonlinear modelling is to regress the linear coefficients of the multiples and powers of inputs to the unknown relationship. Although a simple approach for lower order problems, polynomial approximations do not have good scaling properties and are difficult to implement in hardware, due to the signal saturation effects of analog circuits when the higher order polynomial coefficients tend to be very small. Additionally, high order inputs can result in numerical instabilities while their coefficients are being determined.

The most fundamental polynomial approximation technique involves the Least Squares Method which works such that a matrix of input vectors X (Equation 1) and an output vector Y may be directly used to directly obtain the functional relationship A as shown in Equation 2.

$$X = \left[\ x_1\ x_2\ x_3\ \ldots\right] \tag{1}$$

$$A = \left(X^T Y\right)^{-1}\ X^T X \tag{2}$$

1.2.2 Orthogonal functions

Based on the concept of combining functions as if they ran along a cartesian coordinate axis, orthogonal functions can easily be combined to represent signals that are highly sinusoidal in nature. For nonlinearities that are not as easily decomposed, the necessary (and hopefully anticipated) basis functions are complex and not generally found in lookup tables, increasing the difficulty of online software implementation. A wide range of highly varying functions is also expensive to implement in hardware.

1.2.3 Splines

As piecewise polynomial approximations, they are an improvement in accuracy, but in the end suffer the same problems as polynomials for real time.

1.2.4 Gradient Descent Based Approaches

By assuming that an error surface approximates a parabola, parameters are varied to follow the direction of the rate of change of the error in a performance index function toward the error minimum. The current estimate of each variable coefficient is usually modified by adding the product of the current input value, the performance index error, and a small learning rate constant to slow the rate of convergence and avoid overshooting the optimum.

Via the Lyapunov stability criterion,[1,2] if the derivative of the performance index can be proven always negative, then a local error minimum will be reached in an asymptotically stable fashion. If the local minimum coincides with the desired global minimum, the result has been both stable and accurate. As a first order method of error approximation, it is relatively simple and fast executing. The function approximation methods that follow, along with neural networks (to be discussed later), frequently employ gradient approaches.

1.2.4.1 Least Mean Squares (LMS)

A fully digital approach, LMS is easily implemented in hardware. It assumes that the nonlinear function of interest can be modelled by a system of weighted delay stages. Each delay stage serves as a shift register position or vector element to which a gradient adjusted weight is applied, resulting in an adaptive filter. In one implementation,[3] a 40 dB magnitude noise reduction was achieved in less than 0.01 seconds for a linear model of an acoustic propagation path. Based on the assumption that the residual error in a system can be approximated via a linearly weighted series of time delays, it is still uncertain if LMS will be able to approximate higher order functions any better than other gradient approaches or Simplex optimization for a nonlinear system. The general weight update for the LMS is shown in Equation 3 with W_k being the adjustable weights, μ the learning rate, and ε_k the output error.

$$W_{k+1} = W_k + 2\ \mu\ \epsilon_k\ X_k \quad \textbf{(3)}$$

1.2.4.2 Recursive Least Squares Method (RLSM)[4]

RLSM, and its cousins the Kalman Filter and the Bierman Algorithm, are recursive filters which linearly estimate the next value of a state variable given a computed covariance of the model's error. Their shortcomings stem from the assumption that the model will give a sufficient approximation of the real system across the entire operating range of interest, and that the initial parameter estimate is not too far off from its optimal value, lest the estimation scheme get caught in an undesirable local error minimum. If an unbounded system state is not modelled (or not modelled sufficiently), the error in the estimation can grow without bound and any system controller dependent upon this estimation can become unstable. Sometimes this divergence is overcome by incorporating a forgetting factor that gives greater weight to the newest errors, but at the expense of losing past experience. Additionally, if the target parameter is a fixed bias value, RLSM inputs must often be augmented by an artificial noise signal to continue optimizing based on the rate of change of the error. This noise jitter is often propagated through the filter to the controlled system and may be undesirable. Above third order in complexity, most systems become very difficult to stably approximate.

1.2.4.3 Extended Kalman Filter[4]

Not only do the higher than first order approximations of an Extended Kalman require foreknowledge of the type of function that is being approximated, the additional steps required to make the nonlinear estimations at each update step are time consuming. The simple additions and multiplications of fuzzy logic and neural networks become very competitive computationally to these higher order recursive approximations. Although the Extended Kalman filter has been found to be a more realistic inertial motion approximator to users of virtual reality simulations (measured via a decrease in motion sickness), Mori, et al, in their paper "An Artificial Neural-Net Based Method for Predicting Power System Voltage Harmonics,"[5] found the time prediction capabilities of a simple feedforward neural network with time delayed inputs quite comparable to that of the Kalman filter. Although parameter estimation can be used to continuously update the model for use in an indirect adaptive control scheme, many disturbances occur too quickly to be estimated in a globally asymptotic fashion, as asymptotic stability only assures stability at infinity, with no assurances as to the transient behavior.

1.2.5 Simplex Optimization[3]

The simplex optimization method is a recursive approximation to an unknown function via optimizing the size of a polygon whose number of sides exceeds the number of variable parameters by one. The measure of fit is determined via comparison to a cost function that is calculated from input/output examples. Although superior to the gradient method for optimization along a rough error surface, its result is a linear coefficient for every variable parameter. This is acceptable as a best compromise for the examples trained on, but may not be very accurate across a wider range, if truly accurate for the given examples to begin with. Like self tuning, it can be proven stable and is best implemented when another method has narrowed the error to reasonably small values. The number of variables it can successfully optimize is limited to about five -- beyond that the polynomial sends the lengths of its sides to extremes in an attempt to quickly minimize the error.

1.2.6 Statistical Approximations

Many approaches fall within this category, only some of which will be touched upon here. All are based on the concept that randomness can overcome limitations in error surface negotiation, and that the number of attempts necessary to approximate the coefficients of the assumed component parameters in the representative function is not excessive.

1.2.6.1 Monte Carlo Methods

A phrase that encompasses any method that chooses the next set of variable values via a random number generator, the result of evaluating the performance index for the latest guess is compared to that of the last winner. If no improvement is made, the weights are simply rejected and another random set of weights is tried. Without a gradient like direction to follow, or some other rule of thumb, the effectiveness of the method from one day to the next can vary widely and only general statistical assumptions can be made. A model made via Monte Carlo is better than none at all, and with luck, it will be a good one, if time permits enough trials.

1.2.6.2 Genetic Algorithms[6]

In a commonly implemented genetic algorithm (GA), all variables are arranged into a vector called a chromosome. Individual entries, or genes, in the vector are mutated by varying one increment up or down to create offspring chromosomes. At a randomly chosen breakpoint, a chromosome trades a section of itself with a sibling, approximating the crossover that occurs as a result of mating. All candidate offspring are then evaluated by a performance measure, (i.e., an absolute distance from a target), and are ranked. The best ranked offspring is then compared to every other offspring to yield diversity values, and another ranking is performed. Performance rank and diversity rank are then combined in a user-defined fashion to yield a final rank to which probability values are assigned. Offspring passing on to the next step are chosen with a higher probability from the higher ranks, and a lower probability from the lower ranks. Normally half of the offspring are chosen to continue, maintaining a constant population. Repeat mutation, crossover, and ranking continues until either performance stops improving or an error tolerance is reached.

GAs can very quickly cross moats (i.e., error surfaces with no continuous trend in the optimal direction) in an error surface to find a global optimal point, as long as the search grid of delta values is fine enough, and a large enough population of chromosomes is used. Two drawbacks of the genetic approach are that, due to its randomness, there is no guarantee that an acceptable minimum will be found, and even if the region containing the optimal point is located by the GA, the odds of hitting that optimum are slim. Despite these drawbacks, their desirable advantages make GAs very well suited as a rough first approximation for a final gradient-based optimization for both neural network modelling and control. GAs have been used to swap weight values within a neural network controller which has been held initially stable by a fixed feedback controller, as well as to automatically generate an appropriate neural network via a fractal algorithm.[7]

1.2.7 Harnessing of Parallel Processing Speed and Redundancy

The time limitations of some of the above approaches can be alleviated by implementing them in a parallel computer, but clock cycles are expensive even in parallel, and the limit to an approach is simply raised a little higher. Ironically, parallel processing designers are looking to artificial intelligence methods to overcome some of the challenges of parallel processing itself, such as synchronization, master/slave interaction, and fault tolerance.[8-10]

1.2.8 Artificial Intelligence Approaches

The above methods employ deterministic, gradient approximated, and random approaches to the nonlinear function approximation problem. Other methods, known collectively as artificial intelligence methods, attempt to capture a higher order representation via accumulation of experiences, supervised or unsupervised, hardcoded or learned through experience.

1.2.8.1 Expert Systems (ES)[11,12]

ESs are rule based decision trees whose knowledge is collected from a series of human experts who know what the result of the interaction of various inputs to a system is, but cannot easily describe the rules they operate by explicitly. ESs have many advantages, including ease of knowledge transfer to a computer program, easy to follow logic within the inference engine's decision tree, ease of update and a large capacity for storage of knowledge, flexibility of tree modification and weighting, clear representation of process flow contingencies from start to finish, a tolerance for incomplete information, and the ability to make predictions based on this representation.

Despite their straightforward ease of implementation, ESs are not very adaptive to the generation of new designs.[13] Until a model is made of the human design process, it will be difficult to design an expert system capable of adjusting the weights of its decisions after it has already been created. Additionally, ESs have no means of representing the rules spatially or temporally in relation to other rules within the database, and it is hard to measure the accuracy of a decision made by the expert system based on knowledge of experts whose decisions may have been marginally acceptable in the first place. Fuzzy logic and neural networks may be able to offer assistance when imperfect, imprecise, incomplete, and sometimes uncertain knowledge must be added to a database. The escalating problem that experts for high tech processes are scarce, expensive, and in high demand will trigger much effort in this area.[14,15]

1.2.8.2 Fuzzy Logic (FL)

Fuzzy logic formalizes the idea of a class of real world entities where each entity has a certain participation in the class measurable along a continuous variable.[16-19] This is in contrast to probabilities, which represent a degree of knowledge about real entities and to what degree they are likely to be true for a given value.[20] The concept of fuzzy set theory was first formally introduced in a paper by Lotfi Zadeh in 1965, and one of its offshoots, fuzzy logic control, has matured to the point where a model helicopter can now be controlled by voice alone, and a robot can hit a ball that is falling through a pinball array of pegs with a dart almost every attempt.[21]

By grouping relative measures of an input or output of a system into (usually overlapping) sets, fuzzy logic allows a controller to apply linguistic rules similar to those that a human uses. For instance, if the onions in a frying pan aren't quite brown, but the butter is burning, it is logical to lower the flame a little bit. However, if the desired browning rate of the onions is subsequently not fast enough, increase the flame a tiny bit. Exact values, both incoming and outgoing, are not necessary, but the control algorithm must be fast enough to save the non-stick surface of the pan.

Fuzzy set theory is a tool for anyone who is concerned with uncertainty where the interval of confidence can be defined. The applications of fuzzy sets have been made everywhere and in many practical problems. Through the use of fuzzy sets and fuzzy knowledge, optimization techniques can be developed and used in the solution derived by the system.[22] Also, it is with the use of fuzzy information that large systems can be modelled and analyzed to incorporate the possibility of incomplete, lost, or incorrect information that is being used as inputs by the system.[22,23]

Fuzzy logic control applications are also prevalent and varied. The classic inverted pendulum has appeared often in the literature, where as many as three pendula have been balanced in series.[21] As another balancing control problem,[24] a monorail cart is commanded to balance an inverted pendulum while the cart follows a desired trajectory. Due to the integral inference block that has been added, a step disturbance exists in the angle attained for the pendulum. Balancing has been extended to walking, with promising results. For a biped walking robot[25] with no inverse kinematics model, overlapping gaussian fuzzy membership functions approximate the nonlinear functions of a walking step with continuous derivatives. Fuzzy logic has also proven useful when speed of response is not as important as anticipating a necessary control input based on extremely delayed sensor data, such as for a cement kiln.[26] Adaptive fuzzy systems are already challenging older neural network designs, such as Kosko's adaptive fuzzy logic tractor trailer controller, which outperforms a one hidden layer neural network for backing up to a dock.[27]

Beyond control, the generalization capabilities of fuzzy logic have been repeatedly demonstrated. For planning, fuzzy logic offers a fast, explicit way to implement an expert's imprecise or incomplete knowledge, given that expert systems alone perform best if the information and data is part of a crisp set. Fuzzy logic has been employed in the modelling of uncertainties,[28] selection of optimization strategies,[29] conceptual modelling and cognitive mapping for operations research,[30] the integration of rule based and procedural methods to solve optimization problems through ES technology,[31] dynamic programming,[32] and interactive multiobjective optimization of decisions.[33] If some relationships between inputs and outputs are not apparent to the expert, neural networks can offer a model.[22,23,34] Two illustrative fuzzy logic applications close to this work's implementations are outlined in Appendix A.

The use of fuzzy logic has several advantages over traditional crisp data or logic. Implemented as simple additions, multiplications, and comparisons, FL is easily transferred to real time applications. The linguistic measures of belief can be adjusted until the user is satisfied, and will work effectively at any point during tuning. FL can easily adapt to relationships that are too ambiguous to be captured in equations, and are not susceptible to limitations of equations such as the unstable asymptotes near small numbers and the wraparound of sinusoidal functions. Fuzzy control offers a smooth transition between operating modes, avoiding the possible instabilities found at the boundaries of gain scheduled controllers (such as that between autopilot and human pilot mode in an ice storm). A fuzzy controller, by having explicit reactions that are clearly outlined for the user, is often more accepted by a customer than other forms of AI, especially if online adaptation simply mixes a group of otherwise stable fuzzy outputs. Additionally, fuzzy logic can explicitly provide an equilibrium action, especially important for control problems that require nulling a scalar error measure and maintaining that equilibrium state.

There are some drawbacks to fuzzy logic. Simple analytic controllers will still have a faster response time -- one must consider the total number of operations required per time step, as well as the possible failure modes. FL requires a good conceptual understanding of the relationship between the input and output. If these relationships are not understood in all areas of an implementation, neural networks can give the designer a good rough estimate of the variable dependencies in a given control problem, so that he or she will be able to focus their efforts more intelligently when designing a fuzzy controller. Subsets of rules must be defined continuously over all regions of each universe of discourse, or discontinuous control actions may occur. Further, tuning can only improve so much -- sometimes one can encounter a waterbed effect, where improvements made in one area of the response surface will degrade response in others. Neural network tuning of the shape or degree of overlap of these functions can improve performance. In practice, implementers have had trouble producing intuitive results for relationships with more than two inputs, and must therefore stagger two fuzzy logic controllers to encompass the full problem.

1.2.8.3 Neural Networks (NN)

The computational approach to artificial intelligence (AI) and cognitive engineering has undergone a dramatic evolution over the past few years.[35] This transformation has caused the reasoning methodology to change from discrete symbolic reasoning to massively parallel, connectionist neural modeling.[36] There are two basic types of problems need to be dealt with in the design, development, and application of these distributed computing systems: deterministic and non-deterministic.[35] Deterministic problems are clearly defined and are targeted for solutions that are deterministic in nature, precisely controllable, and can best be handled by computers that employ rigorous, precise logic, algorithms, or production rules. Non-deterministic problems are unstructured and imprecise in nature which lead to computational problems that are inherently ill-posed and ill-conditioned. This leads to decisions being made, often without prior thought. The decisions made are based on information that is incomplete, often ambiguous, plagued with imperfect or inexact knowledge, and involve the handling of large sets of competing constraints that could tolerate close enough solutions.[35]

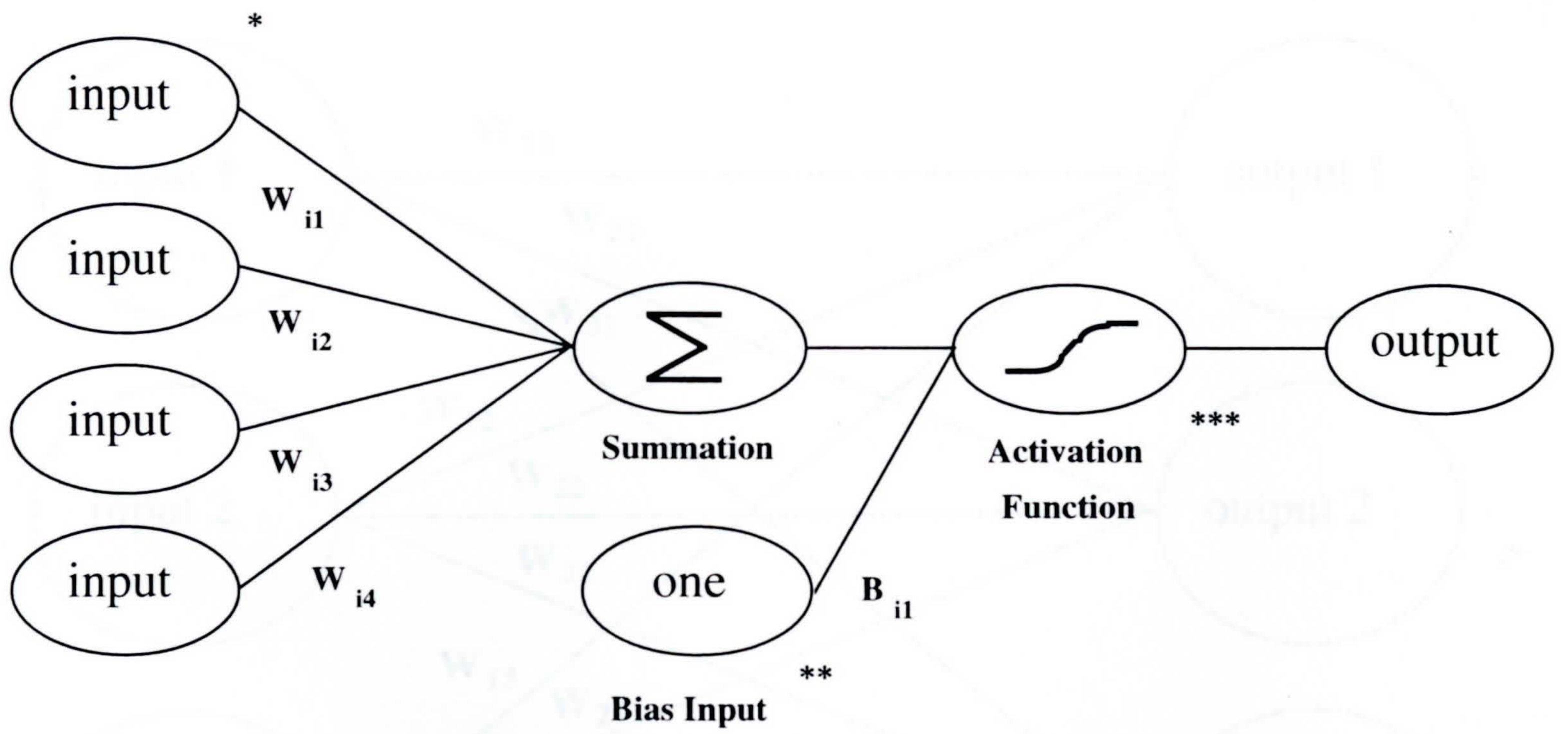
The Perceptron, an early building block of neural nets, was first created by Rosenblatt [21]. He proved, in his Perceptron Convergence Theorem, that if a training set (commonly a matrix of inputs introduced to the net one row at a time) is linearly separable, then a Perceptron will, in a finite number of iterations, learn the function:

$$g_i(input) = input * w_i^T \tag{4}$$

where w_i represent adjustable weights. A perceptron can be pictured as a number of inputs linked to an output with weighted connections which must be trained. Its training technique is called the perceptron training rule, and is used for simple pattern classification where only a linear relation is needed. Perceptrons have a single layer of hard limit neurons which generally give either zero or one as an output.

A neural network is a series of interconnected processing nodes (Figure 2) modelled after those in the human brain that, when introduced to a series of inputs called a training set, can learn a nonlinear relationship between them.[37-39] Figure 3 is an example of a simple neural network. The outputs of a neural network can either be compared to desired outputs and the weights within the network updated to reduce the error (supervised learning), or the network can employ its own internal rules to learn to recognize similar inputs by categorizing them (unsupervised learning). Because of its

Figure 2: A Typical Neural Network Neuron

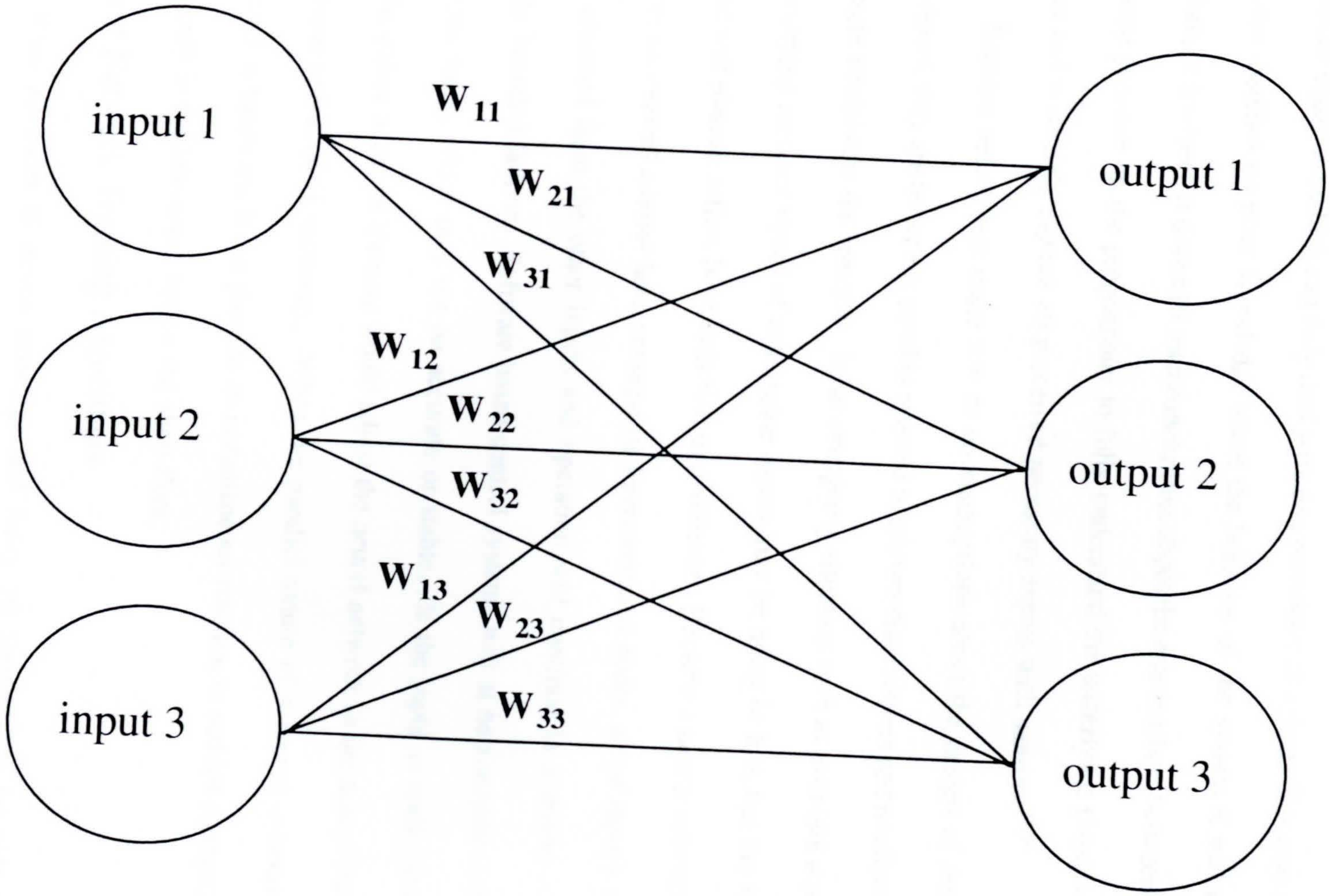


* Inputs are from the external environment or outputs of other neurons.

** Bias input of one is used to displace the activation function in parameter space.

*** Activation functions can be sigmoids, thresholds, linear, sinusoidal, etc.

Figure 3: Basic Neural Network With No Hidden Units



parallel design, a neural network implemented in hardware can reduce computation time by orders of magnitude over ungainly sequential algorithms like the Lagrange-Euler dynamics equations which can only deal with one parameter at a time. Because a neural network assumes no prior knowledge about the behavior of the system it will control, emulate, or interpret, it draws its own conclusions about the relationships between inputs, relieving pressure on the programmer to fully understand the underlying physics of the system and accurately capture all perceived necessary inputs with sensors.[37,38]

Because neural nets make few or no assumptions about the design of the plant they control, they are better equipped to respond to uneven distributions and nonlinearities that might complicate the scenario. In an emergency situation such as avoiding a person while carrying an open vessel of acid, some inputs may be noisy or lost, but the neural net must still plan an action, be it attraction or avoidance. Because a neural network can distribute its internal control laws amongst its interconnected nodes, a lost input's value can be assumed from the other inputs and operation will continue in a stable, if not originally intended fashion. Software based control systems will at best assume a fixed or previous value, which may not be accurate or stable. If the input is really lost or unreliable, online adaptive learning would adjust the neural network to the new situation and continue operation if necessary. When the parallel nature of a neural network is implemented in hardware, it can provide an instantaneous reaction to sudden changes in plans not only in the laboratory but on the shop floor.

1.3 Neural Network Training Algorithms

In time, advances in neural networks and fuzzy set theory may be able to overcome the misconceptions that have barred robots from many possible applications in industry and in the home. Combinations of fuzzy logic and neural networks (called neurofuzzy) are many, and are explored in Appendix B. Alternative neural network update methods and architectures are now discussed, leading to a choice of neurofuzzy architecture for a hierarchy that combines many of the optimization methods of this chapter.

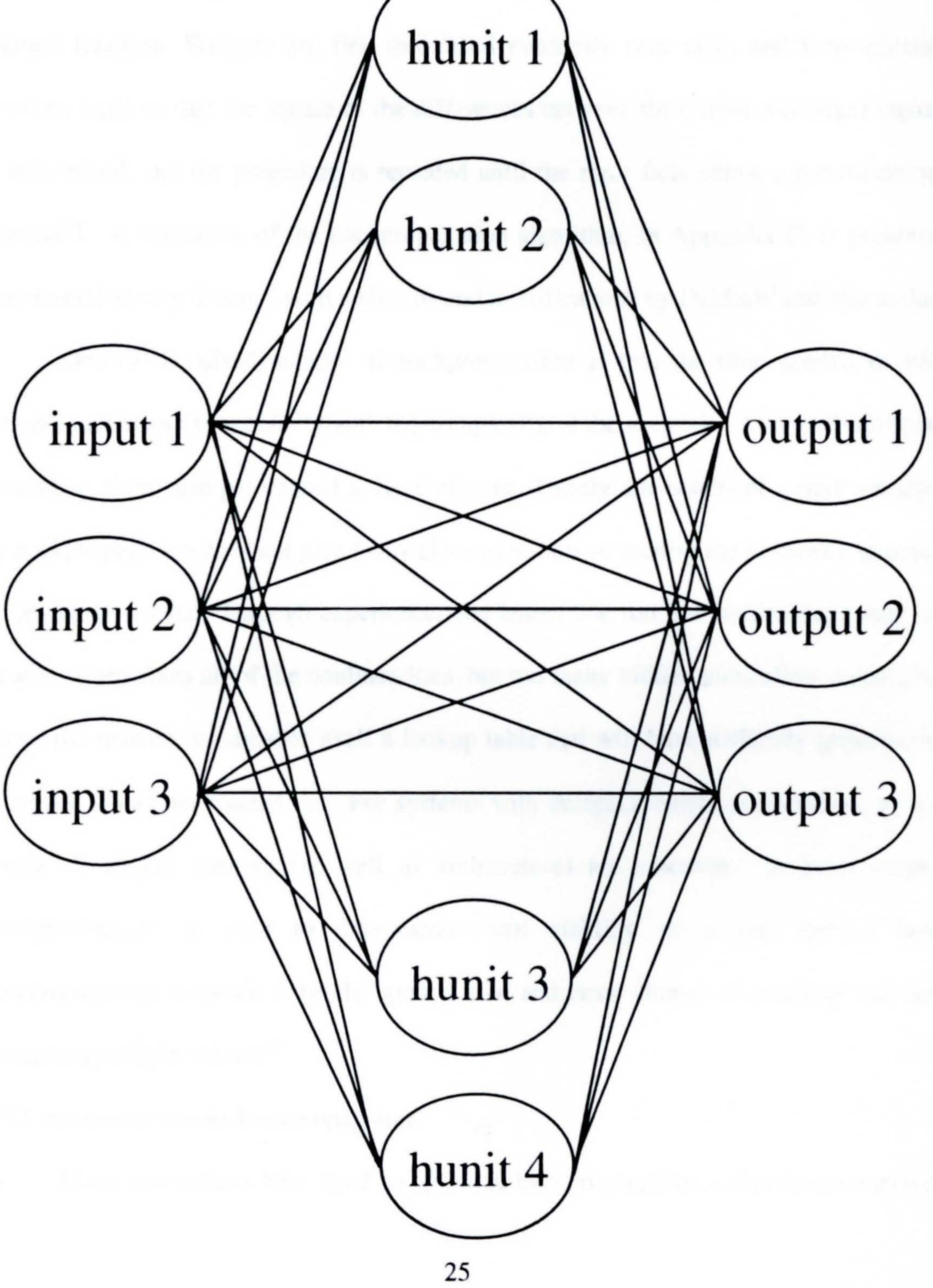
1.3.1 Backpropagation as a Weight Update Method

Backpropagation was created by generalizing the Widrow-Hoff learning rule[40] to multiple layer networks (such as Figure 4) and nonlinear differentiable transfer functions. By minimizing the sum squared errors for each of the training set example outputs with gradient descent optimization, and then distributing incremental weight corrections backward through the layers from the outputs toward the inputs, complex nonlinear functions can be learned in a reasonable amount of time. Eighty to ninety percent of all neural networks today are trained with some variation of backpropagation.

Neural networks trained with backpropagation are good generalizers, that is, any set of inputs never seen before, but within the range of training set inputs, will produce an output that is a smooth interpolation from surrounding points. This property allows them to approximate functions, associate input vectors with specific output vectors (as in vision systems), classify input vectors, steer a car and keep it on the road, or learn robot kinematics and dynamics. Networks with biases, at least one sigmoidal neuron layer, and a linear output neuron layer are capable of approximating any reasonable function.[40]

Each node in backpropagation, like in other neural net update methods, is a

Figure 4: Standard Backpropagation Network With One Hidden Unit Layer



summation of the value of each input, in_i, multiplied by the connection's associated weight, w_{ij}, across all inputs to the node. Often a bias b_i equal to one is added to the summation as an imaginary extra input of one to provide a shift degree of freedom to the learned function. Weights are first initialized randomly near zero, and then adjusted between units so that the square of the differences between the output and target signals is minimized, and the procedure is repeated until the error falls below a predetermined threshold. A derivation of the backpropagation algorithm, in Appendix C, is presented in derived form by Yeung[41] with additions and modifications by DeMuth[1] and this author.

One of the disadvantages of backpropagation is that the time needed to train weights increases exponentially with the complexity of the network. As a gradient based procedure, it can also get trapped in local minima. Thirdly, the results of weight variation by backpropagation does not give many clues as to how to modify the network's structure to get better results. Through experience, one learns that too few hidden units may not be able to represent all of the nonlinearities, but too many hidden units allow overfitting, where the network establishes itself a lookup table that will have difficulty generalizing examples it has not trained on. For systems with complex output possibilities, proper choice of weight training (as well as architecture) are essential. A force control servomechanism is used to demonstrate that stability of a one hidden layer backpropagation network depends upon a trial-and-error choice of learning rate and connection weight values.[42]

1.3.2 Improvements to Backpropagation

Many researchers have tried to optimize backpropagation, either in the methods of updating, or in the structure of the network. One enhancement to standard backpropagation is Paul Werbos' backpropagation through time,[43] which with its additional memory capabilities, allows optimization of dynamic problems such as iterative analysis procedures, neural nets with memory, and time dependent control systems. Its disadvantages, however, include poor forecasts if made far in advance, no compensation for noise, many computations and iterations for training, a continued dependence on gradient search (which can get trapped by local error minima), and lack of real time learning capability.

By making assumptions about a plant or its outputs, various researchers have found faster variations of basic backpropagation. Scott Fahlman[44] claims that his quickprop, loosely related to Newton's method (by assuming that the error is quadratically shaped), consistently outperforms methods such as conjugate gradient. Werbos,[43] at the same time, used Shanno's conjugate gradient method and claimed it outperformed pattern learning methods for a "dense training set made up of distinctly different patterns."

Algorithm enhancement can be attained not only in ways mentioned above, but by more subtle methods, such as building in a priori knowledge of the robot or restructuring the network to suit the level of nonlinearity. Before making extreme changes, one can simply retrain the network with a new set of initial conditions and see if it converges better than the previous network during the initial steep optimization. Weights are often initialized with small random numbers centered around zero, but if one is using a basic two layer network, consisting of sigmoid activation functions in the first, or hidden layer, and linear activation functions in the second, or output layer, then choosing Nguyen-Widrow[45] initial conditions will tend to reduce training times by an order of magnitude.

If the network is assigning weights that are too large, one can reduce step sizes and introduce nonlinear transfer functions into the first hidden layer if they are linear. One can also scale the inputs so that they are similar orders of magnitude and one is not overwhelmed by the influences of the others. If some values range about an average, that bias can be subtracted off of the input training set to utilize both sides of a sigmoid. Networks prefer to see a full variation within their transfer functions. Additionally, a momentum term can be added to Δw_{ij} by multiplying the previous Δw_{ij} by kappa (κ), a positive constant, to prevent the network from being trapped by local minima. This adaptive momentum can be adjusted to be larger early on and smaller later when calculations need to be more precise. Kappa is especially important in networks with many hidden units, where nonlinearities have more freedom to represent the relationships in unintended ways, artificially allowing a local minimum to yield a low error.

Once a network has been found that gives a reasonable answer, the hidden units can be interactively varied for a reasonably small number of iterations to observe which variations minimize error the fastest. When the best net is fully trained, the weights that are almost zero can be eliminated and the network retrained. Simplifications in the number of connections will make the network that much easier to interactively retrain during run mode. Further enhancements to neural network training are discussed in the applications of sections 2.3 and 2.4.

1.4 Neural Network Architectures

The adaptive linear element (ADALINE) was developed by Widrow and Hoff. Unlike the perceptron, the transfer function is linear, not hard limit, so outputs can be continuous instead of discrete. Training utilizes the least mean squares method by adjusting weights and biases according to the magnitude of the error between the current output of the network and its target. Adalines can represent any reasonable function linearly. The network will find as close a solution as possible because the error surface of a linear network is a parabola of as many dimensions as there are linearly independent inputs.

Many architectures fall under the classification of neural network. Some of the most popular are discussed in the next few paragraphs. When considering which architecture is best for an application, one must decide if the states and adaptation parameters are continuous or discrete, if the architecture is set up to accommodate the speed and/or error reduction necessary, if stability of training is critical versus flexibility of options, and finally, what expectations there are for outputs: classification, decisions (go/no go), positive/negative, or discrete/continuous.

1.4.1 Self Organizing Networks (SON)

Neurons in these networks, such as the Instar, Outstar, and Hopfield,[40,46] competitively categorize input vectors on their own without a supervisory comparison to a desired output, and are especially useful in vision-oriented applications such as object recognition and darkness contrast in a variable light environment.[47] SON's have been demonstrated in learning game strategy where a goal function cannot be used to infinite memory requirements, but the algorithm is not very stable.[48]

A popular SON architecture is the adaptive resonance theory network (ART).[49] This architecture autonomously learns to classify arbitrarily many vectors into recognition categories based on predictive successes, maximizing generalization and minimizing predictive error under fast learning conditions, achieving 100% accuracy on a test set when used on a machine learning benchmark database. Each ART network learns to make accurate predictions quickly, efficiently, and flexibly as its stable learning permits continuous new learning, on one or more databases. Learning does not erode prior knowledge until the full memory capacity of the system is exhausted. In one application,[50] a simulated rabbit is trained via an ART network to react to a predator in a one dimensional fashion, by either standing still when the predator does not approach, or by moving right or left based on the direction of predator approach. Once trained, the controller becomes a content-addressable memory, retrieving the next appropriate action from the current state of the rabbit and input from the predator. The "resolution" capability of the rabbit to discern unique transitions from one state to the next is directly proportional: 15 recognition nodes represent 15 transitions, and take about an hour of clock time to train on a Symbolics 3600 series computer, a concern as the problem scales larger and increases in dimensions. The ART is a very useful network as long as one can make use of the categories it creates. An application of its use is demonstrated in Chapter 7.

1.4.2 The Boltzmann Machine[51]

The Boltzmann Machine is a digitally implemented, hardware oriented neural network architecture which has a distributed control mechanism and its neurons can only assume a zero or one value. This eliminates the requirement of the design and implementation of a multiplier, which eliminates the need for any comparison requirements of the system for data manipulation. This digital approach frees up precious memory requirements, allows for prespecified accuracy control, and the decision process is reduced to a lookup table. Although in theory ideal for silicon, implementation is still plagued with problems in the tuning of topology and learning parameters. Efforts have been focused on ways to transfer the gains made in other neural network architecture update methods to the digital domain, with promising results.[52,53]

1.4..3 Radial Basis Functions (RBF)

RBFs are a linear weighted combination of gaussian activation functions. Traditional optimization methods such as least squares or Kalman can be used to determine the weights in its one linear weight layer. Due to the narrow gaussian receptive fields, RBFs offer excellent representation of nonlinearities of up to third order in complexity. Higher orders require prior knowledge of the magnitudes of the input signals and an exponentially large number of gaussians to cover the parameter space. In order to represent higher order nonlinearities with a small number of gaussians, one must allow variation of the mean and standard deviation of each gaussian, raising the complexity of training to that of any other neural network with nonlinear activation functions (threshold, sigmoid, sinusoid, etc.).

1.4.4 Multilayer Feedforward Neural Networks

A feedforward network consists of weighted connections between a large number of simple repeated neurons, and the flow of information propagates forward through the network from inputs through neurons in various layers to outputs. Generally, each neuron in a given layer is fully connected to each input or neuron in the previous layer as well as to each neuron or output in the next layer. Any layer that is not an input or output neuron is defined as a hidden layer neuron. Each neuron either simply multiplies and biases the inputs as a linear neuron, or passes the sum of weighted inputs through a nonlinear activation function before adding the bias weight.

Due to the multiple layers of weights connecting nonlinearities, standard optimization methods are no longer acceptable. It is difficult to control the activity within the hidden layers, especially if there are more than one. The Kolmogorov Mapping Theorem[54] implies that a three layer backpropagation network (which includes one hidden layer) should "perform any continuous mapping defined on a compact set" as well as a net with more layers, but the author and Rumelhart[55] have found that highly nonlinear networks benefit from more layers of fewer units, avoiding the look-up table trap of excessive units per layer. Throughout this work, the three layer network of Kolmogorov will be called a two layer network, as the input layer has no trainable weights in a one hidden unit layer network.

A popular update method, backpropagation,[55] assumes that the error contribution of any neuron input is proportional to its overall contribution to the output, and the amount of correction is determined by gradient descent. Considering this assumption, the method works fairly well, although it is plagued with the same local minimum problem, and is too quick to alter weights that used to provide a good match.[56] Many researchers have approached this problem. The dynamics of a manipulator can be approximated with small neural networks joined by a final layer trained with a least squares method,[57] a steepest descent autotuning method can be employed to vary the shape of the sigmoidal activation function,[58] a genetic algorithm can be used to swap weight values within a neural network controller which has been held initially stable by a fixed feedback controller,[59] and the backpropagation update method itself can be digitized to speed convergence.[60,61] The Levenberg-Marquardt[40] algorithm, a second order variant of backpropagation based on Newton's method, is a good compromise between maintaining speed of computation while avoiding local minima, and is demonstrated in Chapter 3. Other attempts have been made to modify backpropagation to explicitly prove that training will stably approach a minimum, but work is preliminary.[62]

Advantages of feedforward networks are many. Both standard and modified, static and dynamic, gradient methods have found acceptable error minima for many practical classification, modelling, and control applications. They can be used for applications where input magnitudes are not known ahead of time and dimensionality of the input space is high. Because they are broken down into functions with easily computed partial derivatives, computations are straightforward and are not ungainly to scale up in size, making neural nets ideal candidates for parallel computer implementation. Neural nets can deal with just about any level of prior knowledge, from gathering a general relationship from sketchy raw data to smoothing out or simplifying the response of a complete set of fuzzy rules.

Capable of capturing higher order nonlinearities than other optimization methods, neural networks difficulties begin to multiply as their size increases. Convergence slows due to a larger error space with many more local minima. Not only does one weight early on affect more weights in later layers, but more training passes are necessary for the network to adequately capture the nonlinearities without overfitting.[63] Some internal interpretation of input effects on weights can be inferred for smaller networks, but larger networks distribute the representation too completely for a researcher to want to risk drawing conclusions or pruning weights after a training session. A neural network can learn a relationship with extraneous inputs, but it can also memorize some inputs at the expense of learning the proper relationship from other more critical inputs.

1.4.4.1 Recurrent Modification to Feedforward Neural Networks

One popular modification to a feedforward neural network is to feed the outputs of some neurons back into themselves or previous layers as additional inputs. Training becomes more difficult, but these paths, by acting as delays, can improve the network's ability to represent time dependent relationships.[64,65] Attempts have been made to find improved weight optimization algorithms for recurrent networks, such as with a decoupled extended Kalman filter, with promising results.[66] Recurrent networks are applied in Section 2.4.

1.4.4.2 Cerebellar Model Articulation Controller (CMAC)

James Albus's CMAC (Figure 5), which W.T. Miller and others[67] at the University of New Hampshire have implemented on an actual robotic control system, is an associative neural network which assumes local generalization. Inputs which are strongly related to certain outputs are arranged near and given more connections to them, while fewer connections are made to train outputs that are not very related. By alleviating backpropagation's connection blow-up problem, Miller claims convergence an order of magnitude faster. The small model in Figure 5 shows a 20% improvement over Figure 4, but a larger net can see even bigger savings in training and update time. CMAC favors a large number of units per layer to ensure hidden units devoted to nearby inputs, but care is needed to ensure a low error solution when the network is asked to interpolate using an unfamiliar set of inputs.

1.4.4.3 Cascade Correlation Neural Network

Fahlman[44], desiring to speed up neural net training and online update, designed a cascade correlation neural net (Figures 6 and 7) that only added hidden units as needed to approximate the outputs within an acceptable error tolerance. This can increase original training time, but the number of connections will be optimal for later online updating of the net. After a training session starts to level off in his error minimization, Fahlman trains a hidden unit to maximize its correlation with the remaining error, thereby adding a nonlinearity to the net, which is then added to the neural net prior to a new round of training. Note that in Figure 7, a cascade network with two hidden units installed, that each new unit sees a contribution from all previous hidden units as if they were independent inputs. This was intended by Fahlman to adapt to error contributed by all influences in the network, including all previous hidden units. If the magnitude of the inputs varies widely, numerous hidden units will continue to add error in their quest to reduce a combined error.

Figure 5: CMAC Network With Far Connections Disabled

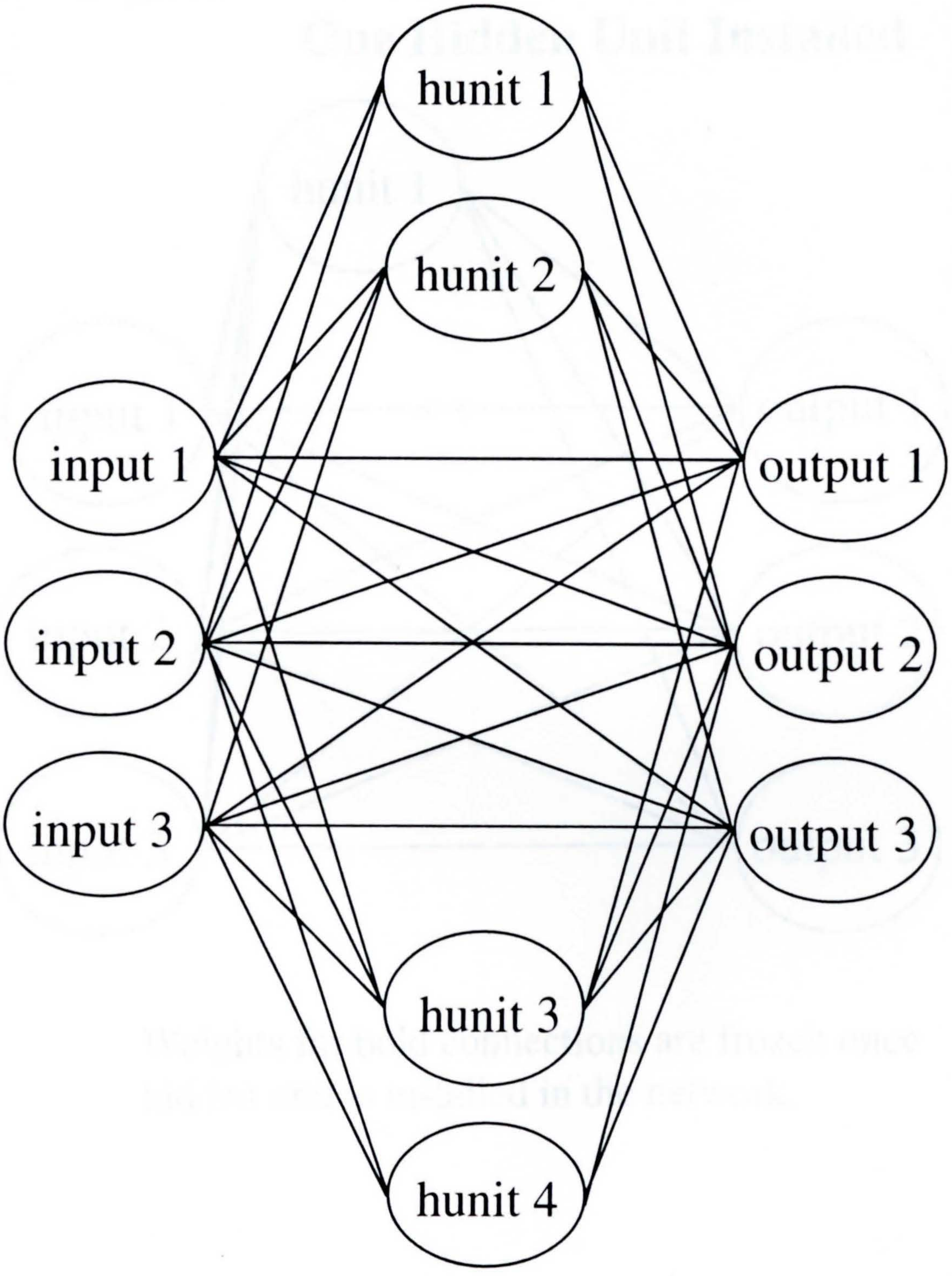
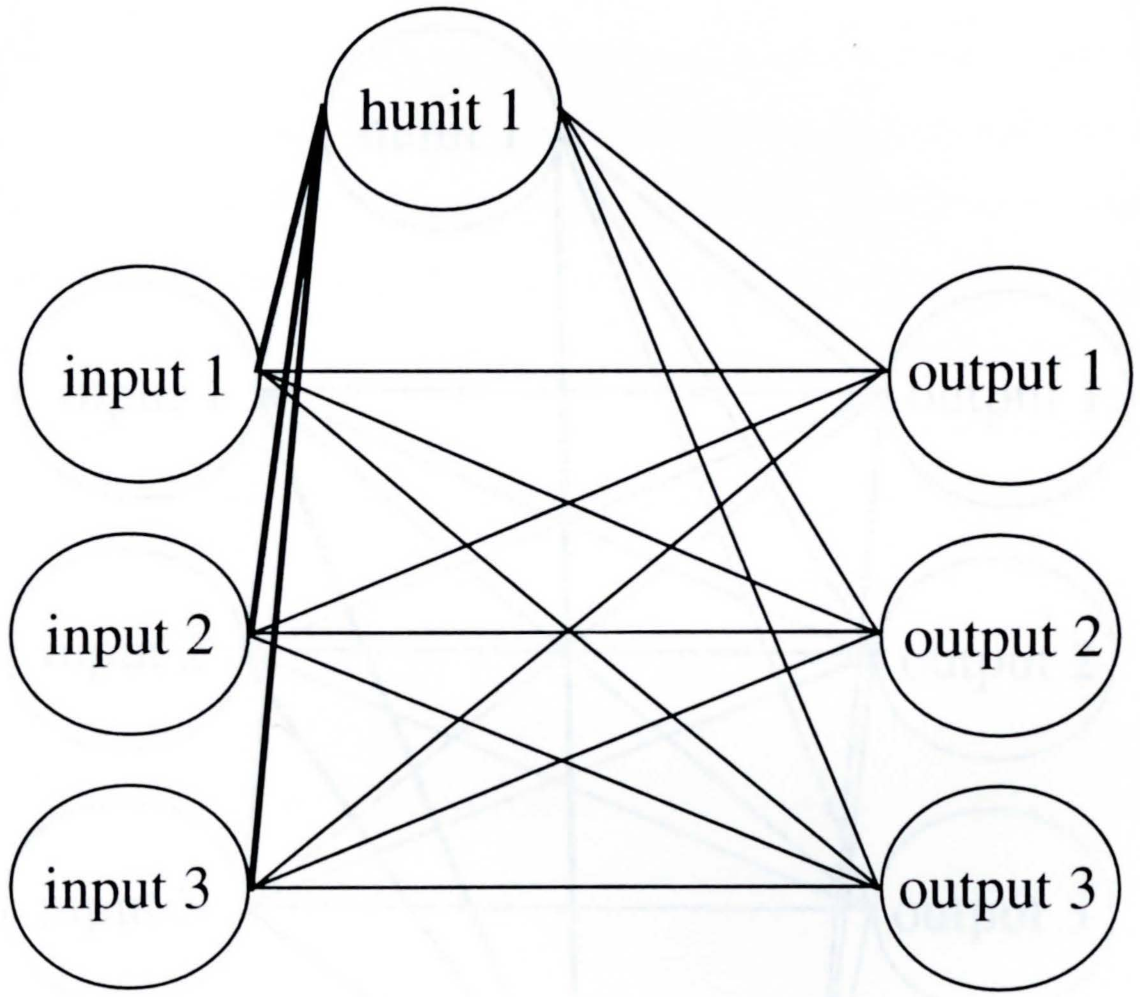
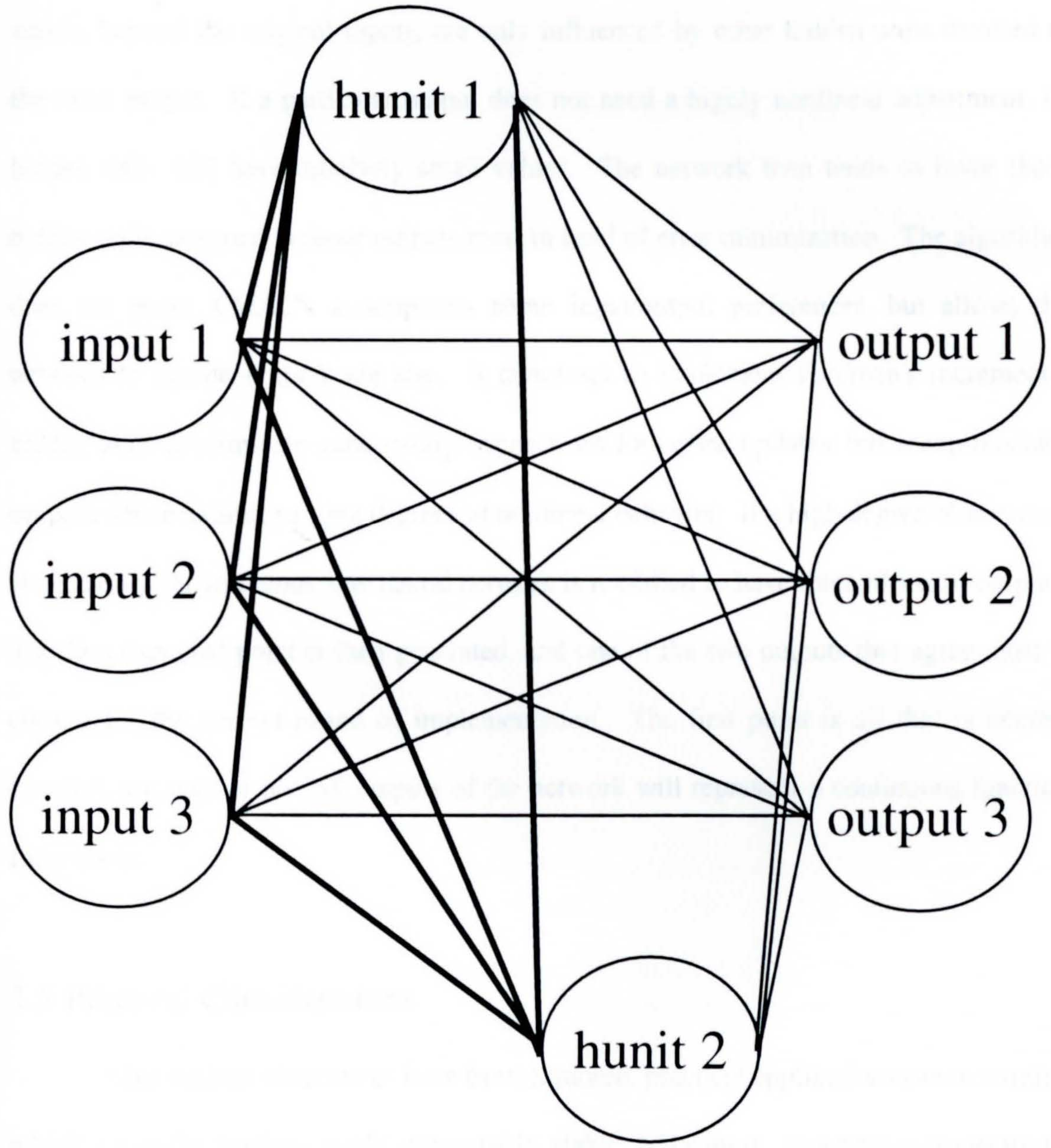


Figure 6: Cascade Network with One Hidden Unit Installed



Weights for bold connections are frozen once hidden unit is installed in the network.

Figure 7: Cascade Network with Two Hidden Units Installed



Weights for bold connections are frozen once hidden unit is installed in the network.

1.4.4.4 An Improved Architecture

An improved architecture for variable range network outputs[68,69] would add hidden units to adapt to errors accumulated, thus approximating a specific output and avoiding needless reiteration. The training algorithm incrementally adds a layer of hidden units which, beyond the original inputs, are only influenced by other hidden units devoted to the same output. If a particular output does not need a highly nonlinear adjustment, its hidden units will have relatively small values. The network then tends to favor those hidden units assigned to those outputs most in need of error minimization. The algorithm does not make CMAC's assumptions about input/output preferences, but allows the network to decide if there are any. It continues to implement Fahlman's incremental hidden units to eliminate unnecessary connections for online updates, but accommodates outputs which exhibit varying degrees of nonlinear behavior. If a high degree of accuracy is necessary for an output, one neural network is modified to have three identical outputs. The first expected point is then generated, and one of the two outputs that agree most is chosen for the current round of implementation. The first point is all that is needed because, accurate or not, all outputs of the network will represent a continuous function from there.

1.5 Practical Considerations

After various alternatives have been explored, practical applications can determine which are easily implementable, expandable, stable, and robust. Practical considerations of implementing the ideas of this chapter are explored in the next chapter, leading to an overall planning and control hierarchy to share information, prioritize and time planning and control responses, and learn from experiences as they are accumulated.

References

[1]Doyle, John C., Bruce A. Francis, and Allen R. Tannenbaum, **Feedback Control Theory**. New York: Macmillan Publishing Company, 1992.

[2]Landau, Yoan D., **Adaptive Control: The Model Reference Approach**. New York: Marcel Dekker, Inc., 1979.

[3]Tascillo, M.A., **An Active Jitter Rejection Technique for High Precision Space Communication Laser Positioning Systems**. PhD Dissertation, Binghamton University, New York, 1994.

[4]Gelb, Arthur, ed., **Applied Optimal Estimation**. Cambridge, Massachusetts: The M.I.T. Press, 1989.

[5]Mori, H., K. Itou, H. Uematsu, and S. Tsuzuki, "An Artificial Neural-Net Based Method for Predicting Power System Voltage Harmonics," **IEEE Transactions on Power Delivery**, Vol. 7, No. 1, January 1992.

[6]Winston, Patrick Henry, **Artificial Intelligence**. New York: Addison-Wesley, 1992.

[7]Vico, F. J. & Sandoval, F., "Use of Generic Algorithms in Neural Networks Definition, " **Neural Networks, Proceedings International Workshop IWANN'91,** Springer - Verlang, Granada, Spain, pp.196-203, 1991.

[8]Dobosiewicz, W. Eskicioglu, M.R., Gburzynski, P., Mutiso, A., "MESS - A Distributed Operating System For The Universe," **Proceedings - Future Trends '90, Second IEEE Workshop on Future Trends of Distributed Computing Systems,** Cairo, Egypt, IEEE Computer Society Press, Los Alamitos, California, pp .208-214, 1990.

[9]Herz, D., Deplanche, A.M., "Fault Tolerance Operators For Distributed Real - Time Control Systems," **Proceedings - Future Trends'90, Second IEEE Workshop on Future Trends of Distributed Computing Systems,** Cairo, Egypt, IEEE Computer Society Press, Los Alamitos, California, pp.113-119, 1990.

[10]Adeli, H. & Kamal, O., "Optimization of Large Structures on Multiprocessor Machines," **Proceedings - Future Trends'90, Second IEEE Workshop on Future Trends of Distributed Computing Systems,** Cairo, Egypt, IEEE Computer Society Press, Los Alamitos, California, pp .290-295, 1990.

[11]Blackburn, M. R., "Using Expert Systems to Construct Formal Specifications," **IEEE Expert,** Vol.4, No.1, pp.62-74, 1989.

[12]Tou, J. T., "Design of Expert Systems For Integrated Production Automation," **Journal of Manufacturing Systems,** Vol.4, No.2, pp.147-156, 1984.

[13]Storey, V. C. & Goldstein, R. C., "Design and Development Of An Expert Database Design System," **International Journal Of Expert Systems,** Vol.3, No.1, pp.31-63, 1990.

[14]Edosomwan, J. A., "Ten Design Rules For Knowledge Based Expert Systems," **Industrial Engineering,** Vol.19, No.8, pp.78-80, 1987.

[15]Srihari, K. & Westby, G., "Expert Systems Development for PCB Assembly Using SMT," **Proceedings - SMTCON '91,** Atlantic City, New Jersey, pp.101-109, 1991.

[16]Baldwin, J.F., "Fuzzy Logic and Its Application To Fuzzy Reasoning," **Advances in Fuzzy Set Theory and Applications,** Edited by Madan M. Gupta, Rammohan K. Ragada, Ronald R. Yager, North-Holland Publishing Co., New York, pp.93-115, 1979.

[17]Tsukamoto, Y., "An Approach To Fuzzy Reasoning Method," **Advances in Fuzzy Set Theory and Applications**, Edited by Madan M. Gupta, Ranunohan K. Ragada, Ronald R. Yager, North-Holland Publishing Co., New York, pp.137-149, 1979.

[18]Negoita, C.V., "Fuzzy Sets In Knowledge Engineering," **The Mathematics of Fuzzy Systems**, Edited by Antinio Di Nola and Aldo G.S. Ventre, Verlag Tuv Reinland Publishing Co., Koln, Germany, pp.265-212, 1986.

[19]Zadeh L.A. "A Theory of Approximate Reasoning," **Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh**, Edited by R. R Yager, S. Ovchinnikov, R. M. Tong, H. T. Nguyen John Wiley & Sons, New York, pp.367-412, 1987.

[20]Wang, P.P., "Fuzzy Set Theory: Past, Present and Future," **Advances in Fuzzy Sets, Possibility Theory, Applications,** Plenum Press, New York, pp.1- 11, 1983.

[21]Schwartz, Daniel G., and George J. Klir, "Fuzzy Logic Flowers in Japan," **IEEE Spectrum**, July 1992.

[22]Zadeh, L.A., 'The Role of Fuzzy Logic in The Management of Uncertainty in Expert Systems', **Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh**, Edited by R.R. Yager, S. Ovchinnikov, R.M. Tong, H.T. Nguyen, John Wiley & Sons, New York, pp.413-441, 1987.

[23]Bellman, R.E. and Zadeh, L.A., "Decision Making in a Fuzzy Environment," **Management Science**, Vol.17, pp.141-164, 1970.

[24]Kawaji, Shigeyasu, and Teruyuki Maeda, "Fuzzy Servo Control System for an Inverted Pendulum," **Fuzzy Engineering toward Human Friendly Systems, Part VII, Fuzzy Logic Control**, IFES 1991.

[25]Ashida, Hitoshi, and Hidetomo Ichihashi, "Fuzzy Learning Control of a Biped Locomotive Robot," **Fuzzy Engineering toward Human Friendly Systems, Part VIII, Application**, IFES 1991.

[26]Maradani, E.H., Ostergaard, J.J., & Lembessis, E., "Use of Fuzzy Logic For Implementing Rule - Based Control of Industrial Processes," **Fuzzy Sets and Decision Analysis,** Edited by H.J. Zimmermann, L.A. Zadeh, B.R. Gaines, Elsevier Science Publishers B.V., Netherlands, pp.429 -445, 1984.

[27]Kong, Seong-Gon and Bart Kosko, "Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems," **Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence**, Englewood Cliffs, NJ: Prentice Hall 1992.

[28]Buckley, J.J., 'Modeling Uncertainty in Operations Research: The Discrete Case', **The Interface Between Artificial Intelligence and Operations Research in Fuzzy Environment,** Verlag TUV Rheinland GmbH, Koln, Germany, pp.13-30. 1989.

[29]Campos, I., "Use of Fuzzy Logic in the Selection of Optimization Strategies," **The Interface Between Artificial Intelligence and Operations Research in Fuzzy Environment,** Verlag TUV Rheinland GmbH, Koln, Germany, pp.51-68, 1989.

[30]Darzentas, J ., "Modelling Fuzzy Knowledge For Soft OR," **The Interface Between Artificial Intelligence and Operations Research in Fuzzy Environment,** Verlag TUV Rheinland GmbH, Koln, Germany, pp.117-126, 1989.

[31]Leung, K.S., Leung, Y., Wong, M-H., "The Integration of Rule-Based and Procedural Methods to Solve Optimization Problems Through Expert-System Technology," **The Interface Between Artificial Intelligence and Operations Research in Fuzzy Environment**, Verlag TUV Rheinland GmbH, Koln, Germany, pp.157-176. 1989.

[32]Esogbue, A.O., Bellman, R.E ., "Fuzzy Dynamic Programming and Its Extensions," **Fuzzy Sets and Decision Analysis,** Edited by H.J. Zimmermann, L.A. Zadeh, B.R. Gaines, Elsevier Science Publishers B.V., Netherlands, pp.67-81, 1984.

[33]Sakawa, M., "Interactive Multiobjective Decision Making by The Fuzzy Sequential Proxy Optimization Technique FSPOT," **Fuzzy Sets and Decision Analysis,** Edited by H.J. Zimmermann, L.A. Zadeh, B.R. Gaines, Elsevier Science Publishers B.V., Netherlands, pp.67 -81, 1984.

[34]Wensley, J.H., "Fault Tolerance in a Local Area Network Used For Industrial Control,"

by choosing initial values of the adaptive weights," **International Joint Conference of Neural Networks**, vol 3, pp. 21-26, July 1990.

[46]Nasrabadi, Nasser M., and Wei Li, "Object Recognition by a Hopfield Neural Network," **IEEE Transactions on Systems , Man and Cybernetics**, Vol. 21, No. 6, November/December 1991.

[47]Lenunon, M. & Vijaya, B.V.K., "Competitively Inhibited Neural Networks: Analysis and Applications," **Neural Networks Concepts Applications, and Implementations,** Vol.III, Editors Antognetti, P. & Milutinovic, V., Prentice Hall, Englewood Cliffs, New Jersey, pp.48-83, 1991.

[48]Bartsev, S. I. & Okhonin, V. A., "Self-Learning Neural Networks Playing Two Coins," **Neurocomputers and Attention II: Connectionism and Neurocomputers,** Editors Arun V. Holden and Vitaly I. Kryukov, Manchester University Press, New York, pp.453-458, 1991.

[49]Carpenter, G.A., Grossberg, S., and Reynolds, J., "A Neural Network Architecture For Fast On -Line Supervised Learning and Pattern Recognition," **Neural Networks for Perception - Human and Machine Perception,** Vol.1, Edited by Harry Wechsler, Academic Press, Inc. San Diego, California, Pp.248 -264, 1992.

[50]Winter, C.L., "Bugs: An Adaptive Critter," **Journal of Neural Network Computing**, Summer 1989.

[51]De Gloria, A. & Ridella, S., "The Boltzmann Machine: Theory and Analysis of the Interconnected Topology," **Neural Networks Concepts, Applications and Implementations,** Vol.II, Editors Antognetti, P. & Milutinovic, V., Prentice Hall, Englewood Cliffs, New Jersey, pp.31-54, 1991.

[52]Caviglia, D.D. Voile, M. & Bisio, G.M., "Analog VLSI Systems For Neural Adaptive Architectures," **Neural Networks Concepts, Applications, and Implementations,** Vol.II, Editors Antognetti, P. & Milutinovic, V., Prentice Hall, Englewood Cliffs, New Jersey, pp.155-206, 1991.

[53]Yair, E. & Gersho, A., "Decision Making and Pattern Classification By Stochastic and Deterministic Neural Networks," **Neural Networks Concepts, Applications, and Implementations,** Vol.III, Editors Antognetti, P. and Milutinovic, V., Prentice Hall, Englewood Cliffs, New Jersey, pp.1 -20.

[54]Lippman, R.P., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, April 1987, p.4-22.

[55]Rumelhart, D.E., G.E. Hinton and R.J. Williams: "Learning Internal Representations by

Error Propagation." **Parallel Distributed Processing**, vol. 1, D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., Cambridge, MA: MIT Press, 1986.

[56]Sutton, R.S., "Two Problems with Backpropagation and Other Steepest descent Learning Procedures for Networks," **Proceedings of the Eight Annual Conference of the Cognitive Science Society**, 1989.

[57]Guez, A., Z. Ahmad, and J. Selinsky, "The Application of Neural Networks to Robotics," **Neural Networks: Current Applications,** edited by P.G.J. Lisboa, New York: Chapman & Hall, 1992.

[58]Yamada, Takayuki, and Tetsuro Yabuta, "Neural Network Controller Using Autotuning Method for Nonlinear Functions," **IEEE Transactions on Neural Networks**, Vol. 3, No. 4, July 1992.

[59]Ichikawa, Yoshiaki and Toshiyuki Sawa, "Neural Network Application for Direct Feedback Controllers," **IEEE Transactions on Neural Networks**, Vol. 3, No. 2, March 1992.

[60]Van Den Bout, D.E., Miller, T.K., & Wike, W.B., "Stochastic Architectures For Recurrent Neural Networks," **Neural Networks Concepts, Applications, and Implementations,** Vol.III, Editors Antognetti, P. & Milutinovic, V., Prentice Hall, Englewood Cliffs, New Jersey, pp.155 -206, 1991.

[61]Stankovic, S.S. & Milosavljevic, M.M, "Training of Multilayer Perceptions by Stochastic Approximation," **Neural Networks Concepts, Applications, and Implementations,** Vol.IV, Editors Antognetti, P. & Milutinovic, V., Prentice Hall, Englewood Cliffs, New Jersey, pp .201 -239, 1991.

[62]Shulman, E.I., "Learning In The Fully Connected Network," **Neurocomputers and Attention Volume II: Connectionism and neurocomputers,** Editors Arun V. Holden and Vitaly I. Kryukov, Manchester University Press, New York, pp.453-458, 1991.

[63]Caudill, M. and C. Butler: **Understanding Neural Networks, Volume 2: Advanced Networks**. The MIT Press, Cambridge, MA, 1992

[64]Jordan, M., and R. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm," in press, **Neural Computation**.

[65]Fernandez de Canete, J., Ollero, A., Diaz - Fondon, M., "Autonomous Controller Tuning by Using A Neural Network," **Neural Networks Proceedings - International Workshop IWANN'91,** Springer-Verlang, Granada Spain, pp.445-452, 1991.

[66]Puskorius, G.V. and L.A. Feldkamp, "Recurrent Network Training with the Decoupled

Extended Kalman Filter Algorithm," **SPIE: Science of Artificial Neural Networks**, Vol. 1710, 1992.

[67]Miller, W.T., F.H. Glantz, and L.G. Kraft, "CMAC: An Associative Neural Network Alternative to Backpropagation," **Proceedings of the IEEE**, Vol. 78, No. 10, October 1990.

[68]Sadeghi, T., M. Tascillo, A. Simons, and K. Lai, "Adaptive Reconfigurable Flight Controls for High Angle of Attack Agility," Proceedings of the North Atlantic Treaty Organization Advisory Group for Aerospace Research and Development, Guidance and Control Panel, Stability in Aerospace Systems, Toulouse, France, June 23-26, 1992.

[69]Simons, A., **Robotic Trajectory Control Employing Anyanet Neural Architecture**, Master's Thesis: Rensselaer Polytechnic Institute, Troy, New York, 1992.

CHAPTER 2:

PRACTICAL MODELLING FOR A CONTROL AND PLANNING HIERARCHY

2.1 Modelling Real World Nonlinear Systems

Some practical considerations of designing a neural network to capture nonlinear relationships in a real world environment are explored, from learning the kinematics of a five axis manipulator, to fitting the curve with linearly dependent inputs, to modelling the dynamics of a automobile engine driven by a dynamometer. The author's contribution in this area is the FSFER network, a modified recurrent network, which excels in its ability to model a time varying nonlinear system in both the time and frequency domains.[1] The FSFER network, with the author's enhancements to weight training for speed and reliability of error minimization, is incorporated into a novel coordinated hierarchy for redundant robot planning and control.

2.2 Examples of Neural Trajectory Training

2.2.1 Robot Inverse Kinematics

The improved architecture was employed to learn the inverse kinematics of the five axis Rhino XR-2 robot (Figures 1 and 2), with the three cartesian points in space plus q_{234} and the independent q_5 as inputs. The joint angles q_1, q_2, q_3, and q_4 were the output angles to train to. The effectiveness of training was demonstrated by inserting the neural networks suggested q angles for a given trajectory into forward kinematics equations, and then plotting the resulting actual trajectory. The resulting plots (Figures 3 through 8, odd numbers labelling the target, and even numbers the resulting trajectory for a selected training set of points) showed that, although the exact location of the trajectory in cartesian space was not attained, the desired shape and orientation was. Demonstration on an actual Rhino robot showed that the shape had been learned, but twisted in cartesian space. The error optimization seemed trapped in local minima.

2.2.2 Weight Training Enhancement

In order to overcome backpropagation's difficulty in avoiding local minima, the modified algorithm preconditioned the trainable weights and biases by arranging them in a vector, varying and evaluating them via a quality measure. The winning vector was then introduced or returned to the network for further optimization. This method, based upon the genetic algorithm concept, excels when desirable performance criterion values are surrounded by undesirable values, a trap for a gradient-based algorithm. The organized randomness of the genetic algorithm jumps over this "moat" by utilizing the concepts of mutation and crossover.[2] Such discontinuous moves are possible by choosing offspring based on diversity as well as optimality.

In order to overcome backpropagation's difficulty in avoiding local minima, the algorithm interrupted training every few hundred iterations to arrange the trainable

Figure 1: Denavit-Hartenberg Link Coordinates For the Rhino XR-2 Robot

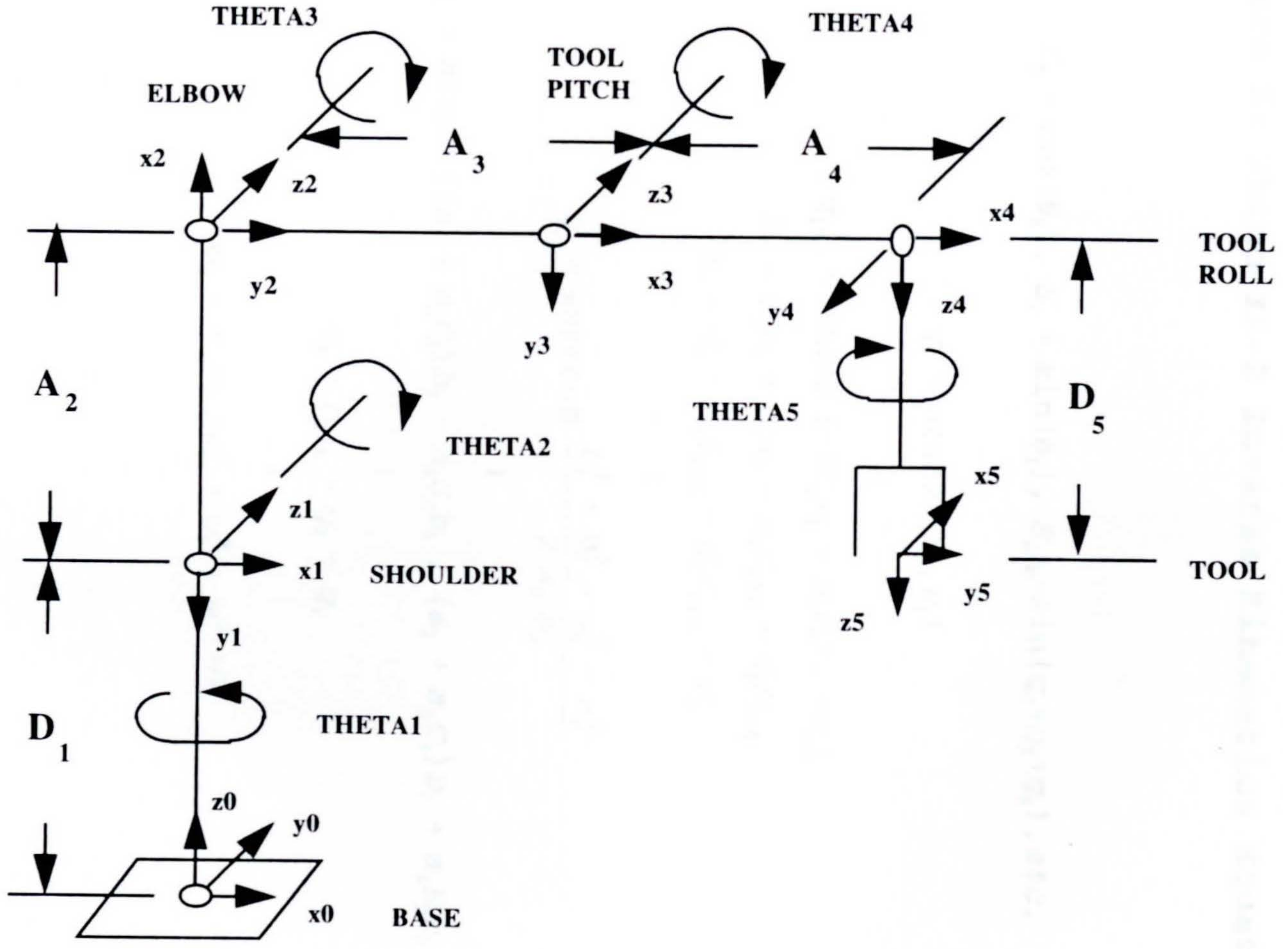


Figure 2: Rhino XR-2 Inverse Kinematics Equations

$$C_1 = \cos(\theta_1),\ S_1 = \sin(\theta_1),\ S_{234} = \sin(q_2+q_3+q_4), etc.$$

$$\downarrow$$

$$q_1 = atan2\ (w_2, w_1)$$

$$\downarrow$$

$$q_{234} = atan2\ [-(C_1 w_4 + S_1 w_5),\ -w_6]$$

$$b_1 = C_1 w_1 + S_1 w_2 - a_4 C_{234} + d_5 S_{234}$$

$$b_2 = d_1 - a_4 S_{234} - d_5 C_{234} - w_3$$

$$\downarrow$$

$$q_3 = \pm\arccos \frac{b_1^2 + b_2^2 - a_2^2 - a_3^2}{2\ a_2\ a_3}$$

$$\downarrow$$

$$q_2 = atan2\ [(a_2 + a_3 C_3) b_2 - a_3 S_3 b_1\ ,\ (a_2 + a_3 C_3) b_1 + a_3 S_3 b_2]$$

$$\downarrow$$

$$q_4 = q_{234} - q_2 - q_3$$

$$\downarrow$$

$$q_5 = \pi\ \ln\ (w_4^2 + w_5^2 + w_6^2)^{1/2}$$

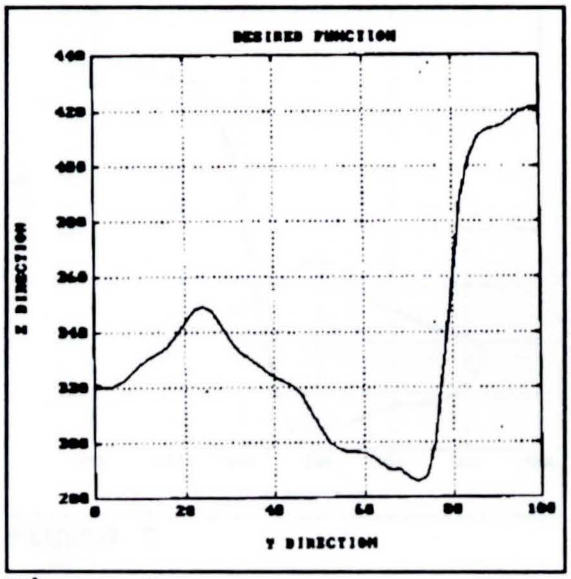


Figure 3

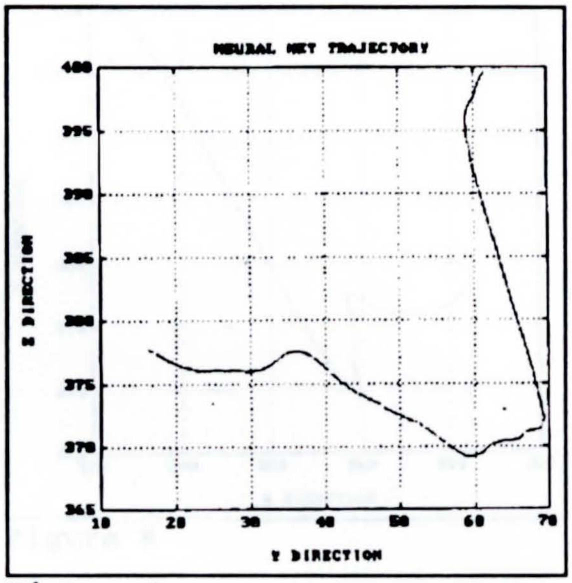


Figure 4

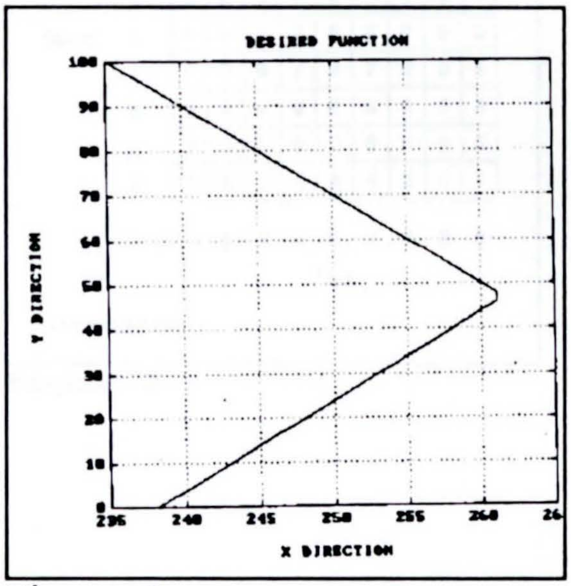


Figure 5

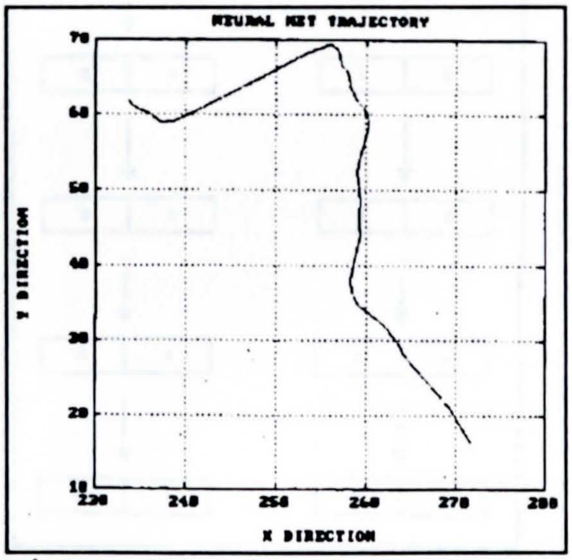


Figure 6

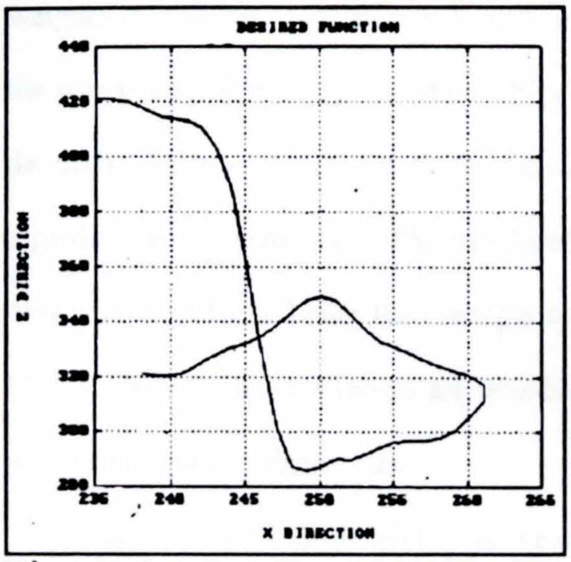


Figure 7

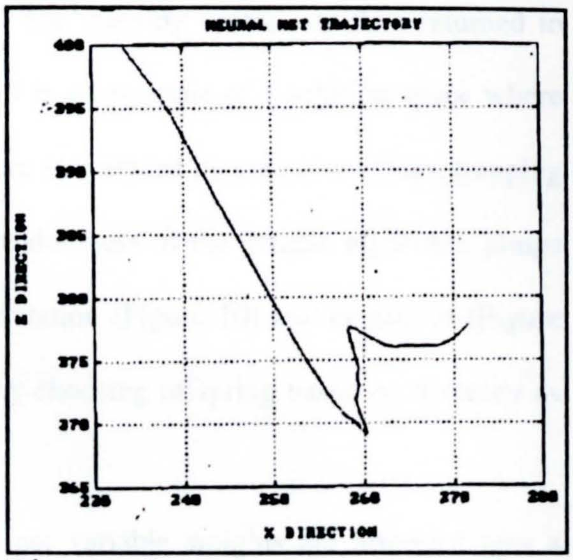
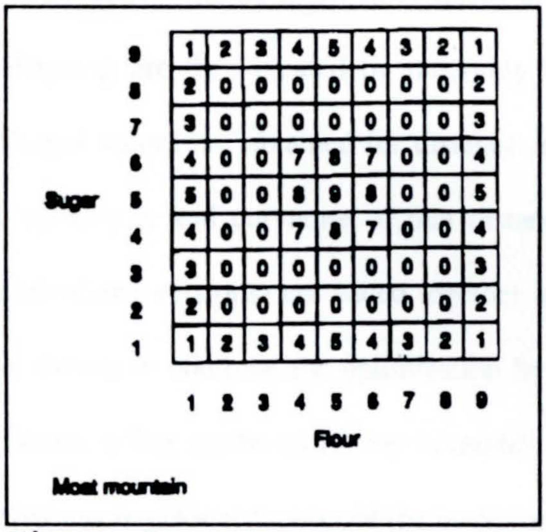


Figure 8



Sugar									
9	1	2	3	4	5	4	3	2	1
8	2	0	0	0	0	0	0	0	2
7	3	0	0	0	0	0	0	0	3
6	4	0	0	7	8	7	0	0	4
5	5	0	0	8	9	8	0	0	5
4	4	0	0	7	8	7	0	0	4
3	3	0	0	0	0	0	0	0	3
2	2	0	0	0	0	0	0	0	2
1	1	2	3	4	5	4	3	2	1
	1	2	3	4	5	6	7	8	9

Flour

Most mountain

Figure 9

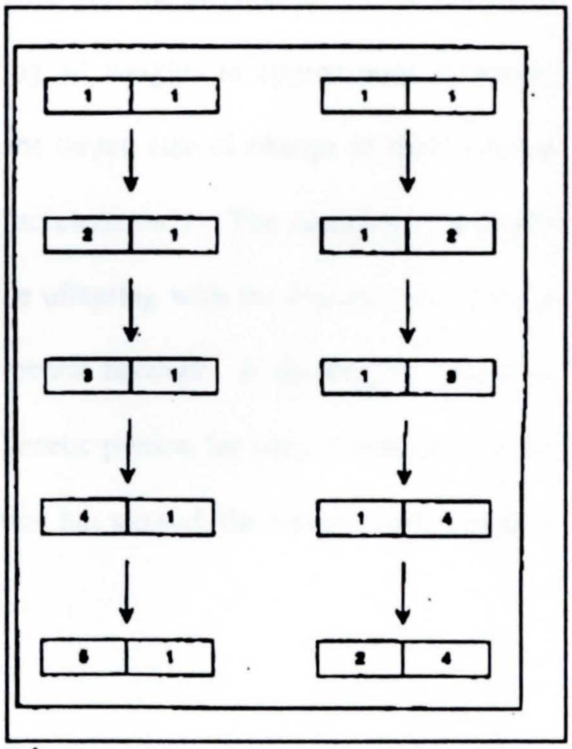


Figure 10

weights in a vector, vary and evaluate them. The winning vector was then returned to the network for further optimization. Figure 9 is an example of a solution space where the desirable values (highest in this instance) are surrounded by very low values, trapping a gradient based algorithm. The structured randomness of the genetic algorithm jumps over this moat by utilizing the concepts of mutation (Figure 10) and crossover (Figure 11). Such discontinuous moves are possible by choosing offspring based on diversity as well as optimality (Figure 12).

In the modified algorithm, all neural net variable weights are arranged into a vector. One weight is randomly varied, creating an offspring of the original vector to approximate mutation. Variation of one weight is continued across the length of the vector until there are as many offspring as weights, plus the original parent. A few more offspring are then created by randomly varying all weights to approximate diversity. Output values are obtained for distance from the target, rate of change of that distance ("velocity"), and curvature of that distance ("acceleration"). The resulting values (or normalized versions) are added together and the offspring with the lowest overall score is chosen to continue for optimization by the neural network. If the original parent is chosen, a flag can be set to stop or modify the genetic portion for later. Otherwise, when progress in error reduction of the backpropagation has slowed, the weights will return to the genetic portion.

2.2.3 Comparison of Approaches

The function in Figure 13,

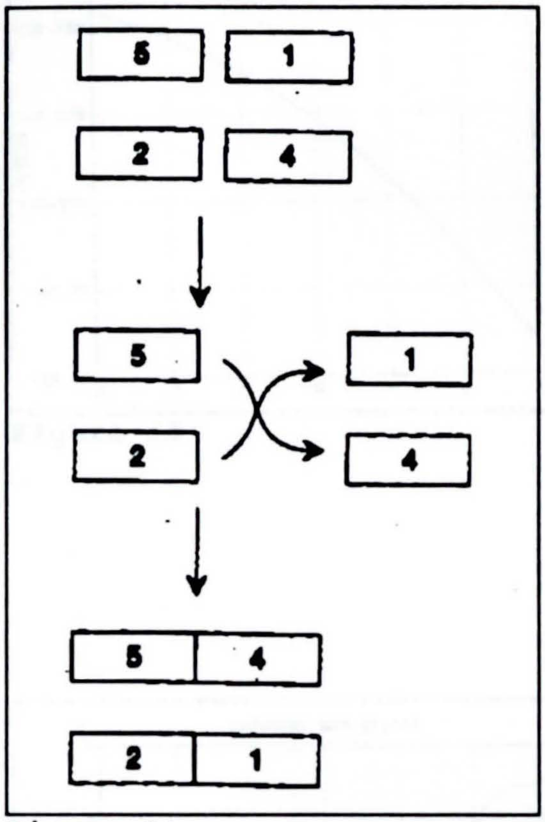


Figure 11

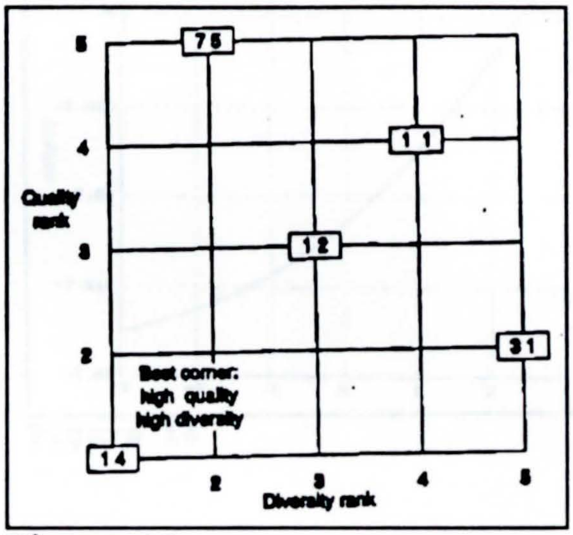


Figure 12

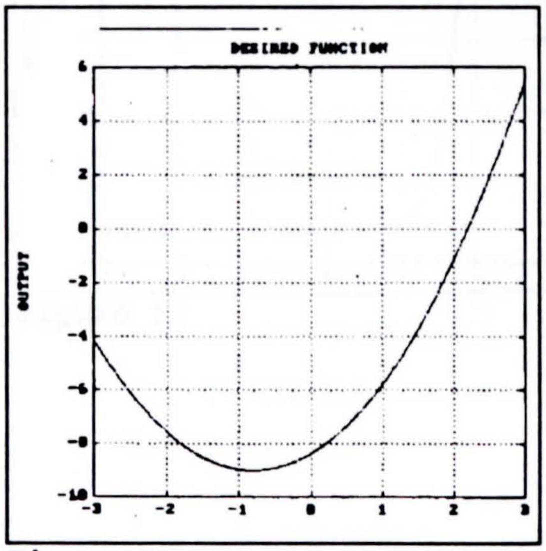


Figure 13

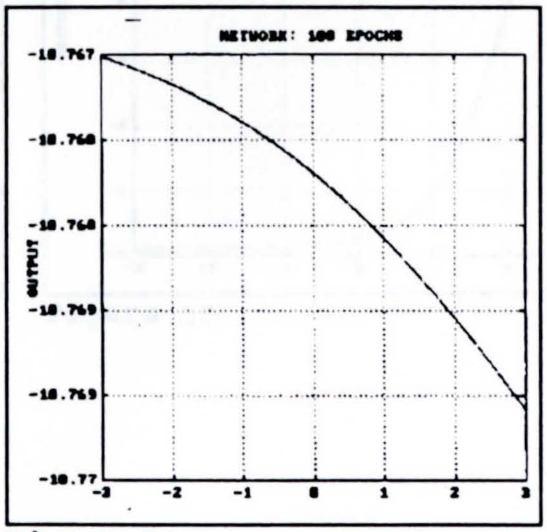


Figure 14

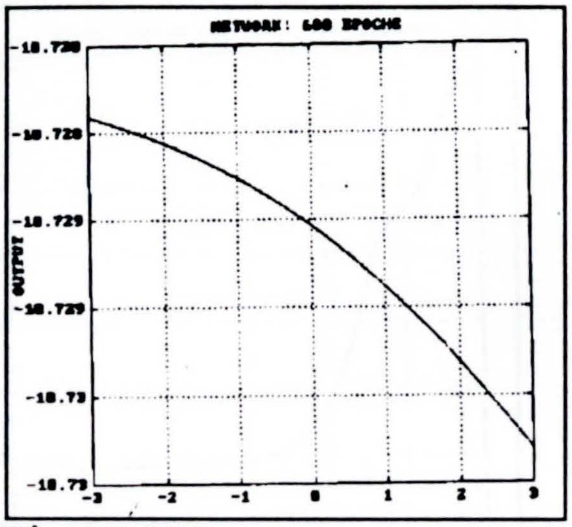


Figure 15

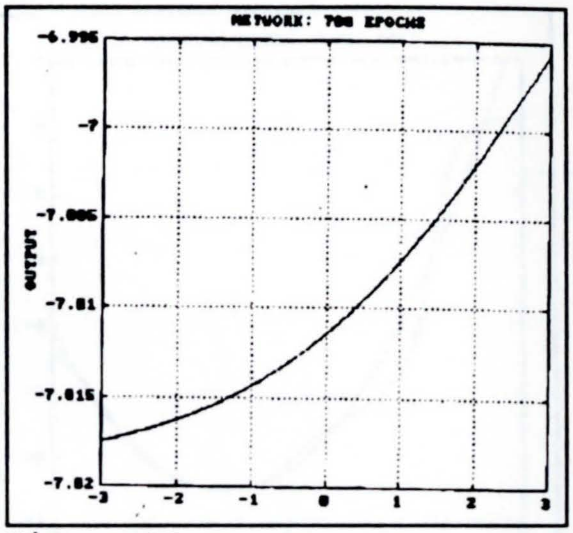


Figure 16

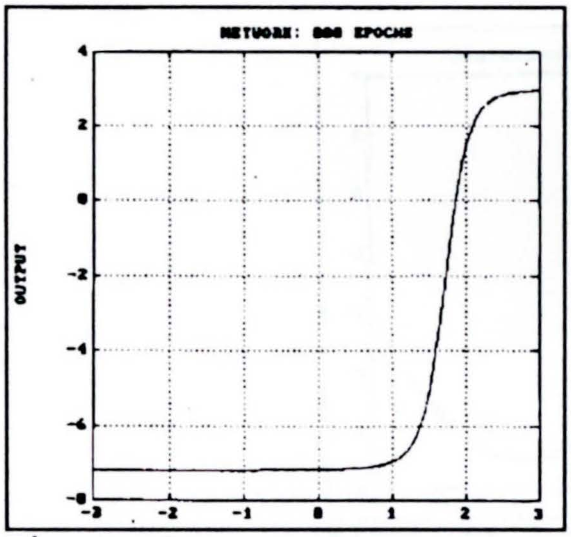


Figure 17

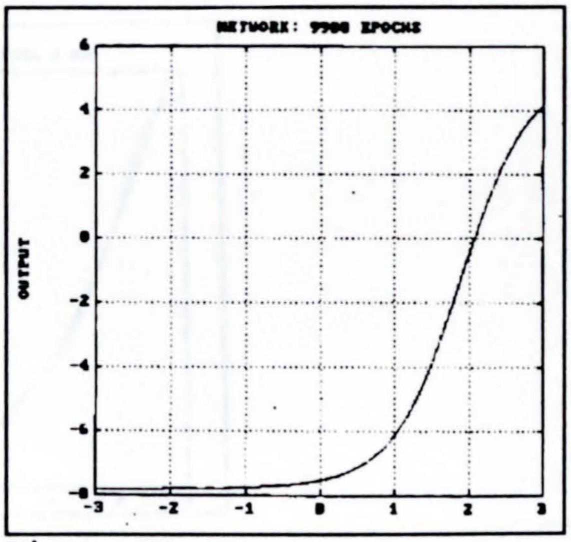


Figure 18

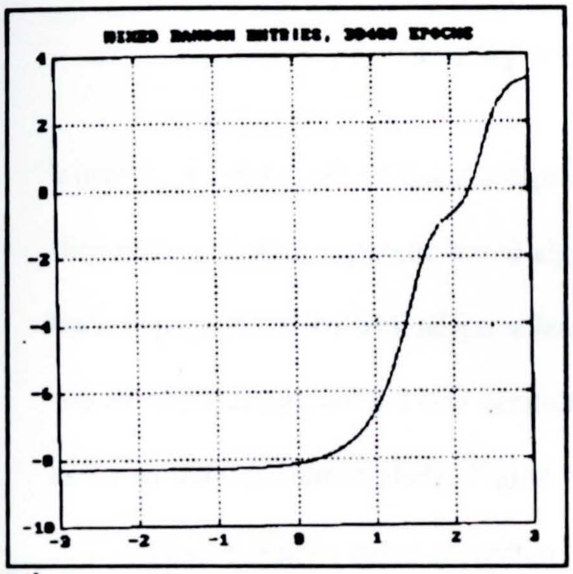


Figure 19

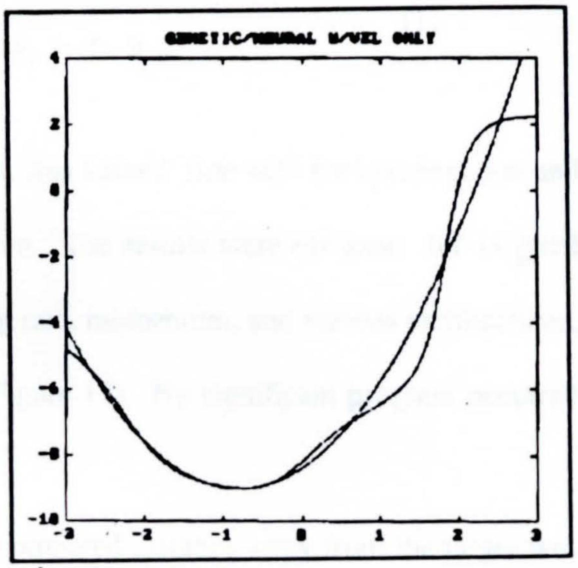


Figure 20

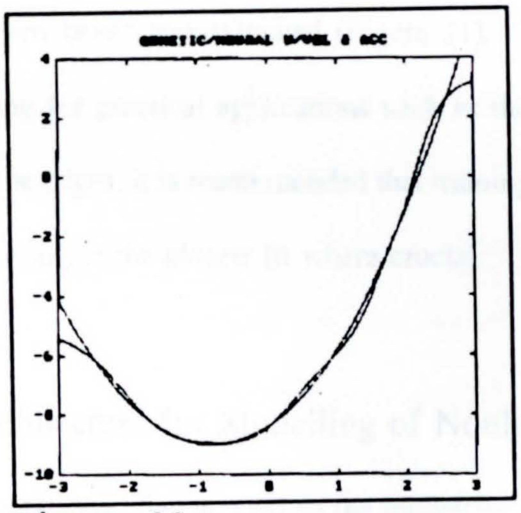


Figure 21

$$y = x_1^2 + 4.0\,x_2 - 0.2\,x_3 - 0.3\,, \tag{1}$$

where x_1, x_2, and x_3 are all linearly dependent, was trained first with backpropagation and then enhanced with a genetic based algorithm. The results were obvious: the original backpropagation, even with adaptive learning rate, momentum, and various architectures, would stop learning around 800 iterations (Figure 17). No significant progress occurred as far as 9900 iterations along (Figure 18).

First, a genetic based algorithm that compared distance away from the target was employed. The curve was better approximated for the positive x axis, but remained flat for the negative x axis (Figure 19). With a rate of change ("velocity") factor added into the performance measure, the negative side lifted up (Figure 20), and with acceleration added, the curve was even better approximated (Figure 21). The genetic algorithm is surely worth investigating for practical applications such as the robotic hand. Because the estimation strays at the edges, it is recommended that training be extended beyond the expected operation range to get the closest fit where crucial.

2.3 A Recurrent Architecture for Modelling of Nonlinear Systems

The above concepts were then applied to the modelling of an automotive engine and dynamometer testbed so that alternative control methods for the noisy torque control loop could be tested offline, resulting in further enhancements to architecture and weight update. An application was found that centered around a popular traditional feedback PID control configuration.[3] Such controllers are conceptually simple and easily implemented, but require routine adjustment as test conditions and engine characteristics vary from test to test. This adjustment does not necessarily result in the best possible performance, especially for transient type testing in which the inherent system nonlinearities are most apparent. As the system at hand is highly nonlinear, and subject to environmental conditions, neural networks are an excellent choice for both on and off line characterization of the process.

2.3.1 The Application Scenario

This application involves the modeling of an engine dynamometer testbed consisting of a heavy duty gasoline engine, DC dynamometer and the associated feedback control mechanism. The principal goal of the work is to obtain a greater understanding of the system and achieve a functional simulation model for the development of an improved throttle control mechanism. All too often, such feedback control designs start with a simple linear model which, although convenient for algorithm development, are typically an inadequate representation of the nonlinear dynamic system. An alternative approach worth considering is to start with a reliable nonlinear model, and then begin control design around linearized portions of that model. This permits finer characterization of the validity of each linear model, and offers an offline evaluation platform for any developed controller. The latter is especially critical in the present application, as the integration and testing of a new controller in any active production facility is disruptive and costly.

For this task, three different neural architectures are considered, standard feedforward,[4] Elman type recurrent,[5] and finally a Full State Feedback Extended Recurrent (FSFER) architecture. Fundamental benefits of the recurrent architecture are demonstrated as well as an evaluation of different recurrent implementations. Neural networks offer a favorable tradeoff between explicit models (which render each component of the system in a very accurate manner but are very specific to a particular system and are time consuming to assemble and execute) and simplified linear representations which are easy to build and use at the expense of validity. For this work, three neural architectures are considered: a standard feedforward, a basic recurrent (Elman network), and finally, a FSFER architecture. These are shown in Figures 22-24 and described in formula form below. Each of the neural networks can be represented in a common matrix form. Represented as $In(\tau) = [in(\tau)\ \ in(\tau\text{-}1)\ \ in(\tau\text{-}2)$ and/or time variants of other signals]', the input vector can either be obtained as a row from a training set during training, or as the last signals obtained from the system once on line. 3.3.1.1 The Feedforward Network

The feedforward network found to be optimal for the problem consisted of two sigmoidal layers, 4 and 8 units apiece, followed by a linear neuron representing its own layer. Therefore, UL1 (Units in Layer 1) will be 4, UL2 will be 8, and UL3 will be 1. NL (Number of Inputs) will equal 5. In the first weight layer, the output vector Out_1 will become:

$$Out_1(\tau) = F_1(W_1 * In(\tau) + B_1), \text{ where} \qquad (2)$$

$In(\tau)$ is a NIx1 vector,

$Out_1(\tau)$ is a UL1x1 vector,

F_1 is a hyperbolic tangent sigmoid activation function,

W_1 is a UL1xNI matrix, and

B_1 is a UL1x1 vector.

In the second weight layer:

$$Out_2(\tau) = F_2(W_2{*}Out_1(\tau)+B_2), \text{ where} \qquad (3)$$

$Out_2(\tau)$ is a UL2x1 vector,

F_2 is a hyperbolic tangent sigmoid activation function,

W_2 is a UL2xUL1 matrix, and

B_2 is a UL2x1 vector.

In the third weight layer:

$$u(\tau) = F_3(W_3{*}Out_2(\tau)+B_3), \text{ where} \qquad (4)$$

$u(\tau)$, the output of the neural network, is a scalar,

F_3 is a linear activation function,

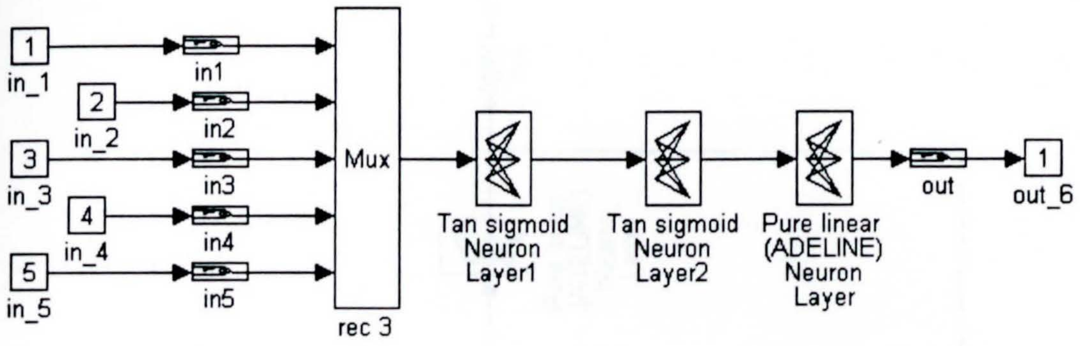
W_3 is a UL3xUL2 vector, and

B_3 is a scalar (or UL3x1).

The complete Feedforward function then becomes:

$$u(\tau) = F_3(W_3{*}(F_2(W_2{*}(F_1(W_1{*}In(\tau)+B_1))+B_2))+B_3). \qquad (5)$$

STANDARD FEEDFORWARD NETWORK



1
in_1
2
in_2
3
in_3
4
in_4
5
in_5
in1
in2
in3
in4
in5
Mux
rec 3
Tan sigmoid Neuron Layer1
Tan sigmoid Neuron Layer2
Pure linear (ADELINE) Neuron Layer
out
1
out_6

Figure 22

ELMAN NETWORK

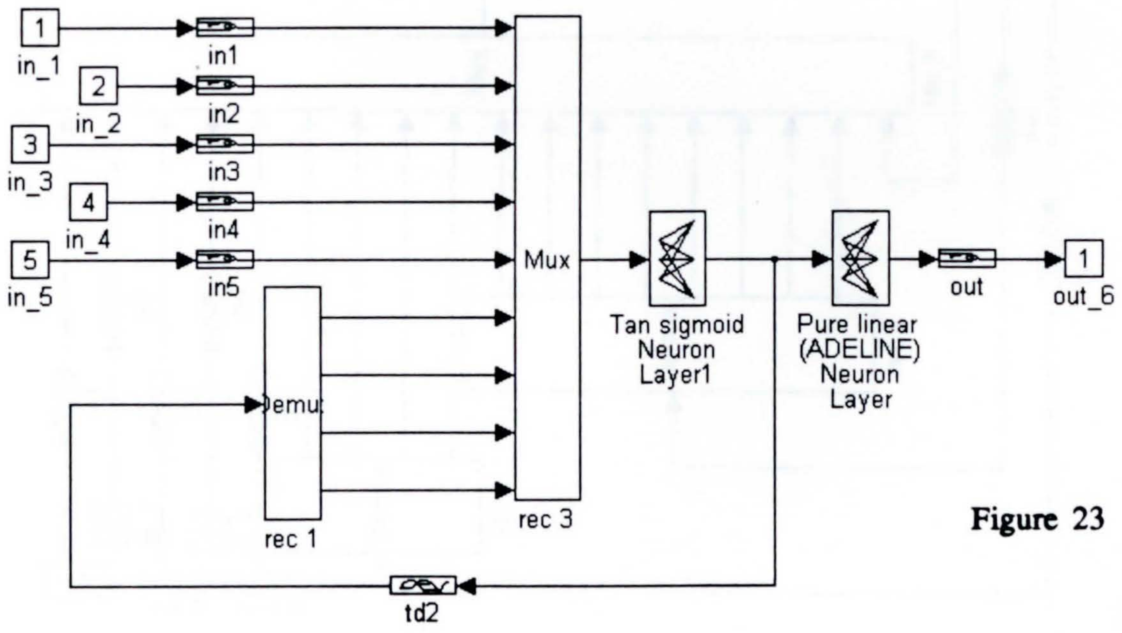


Figure 23

FSFER NETWORK

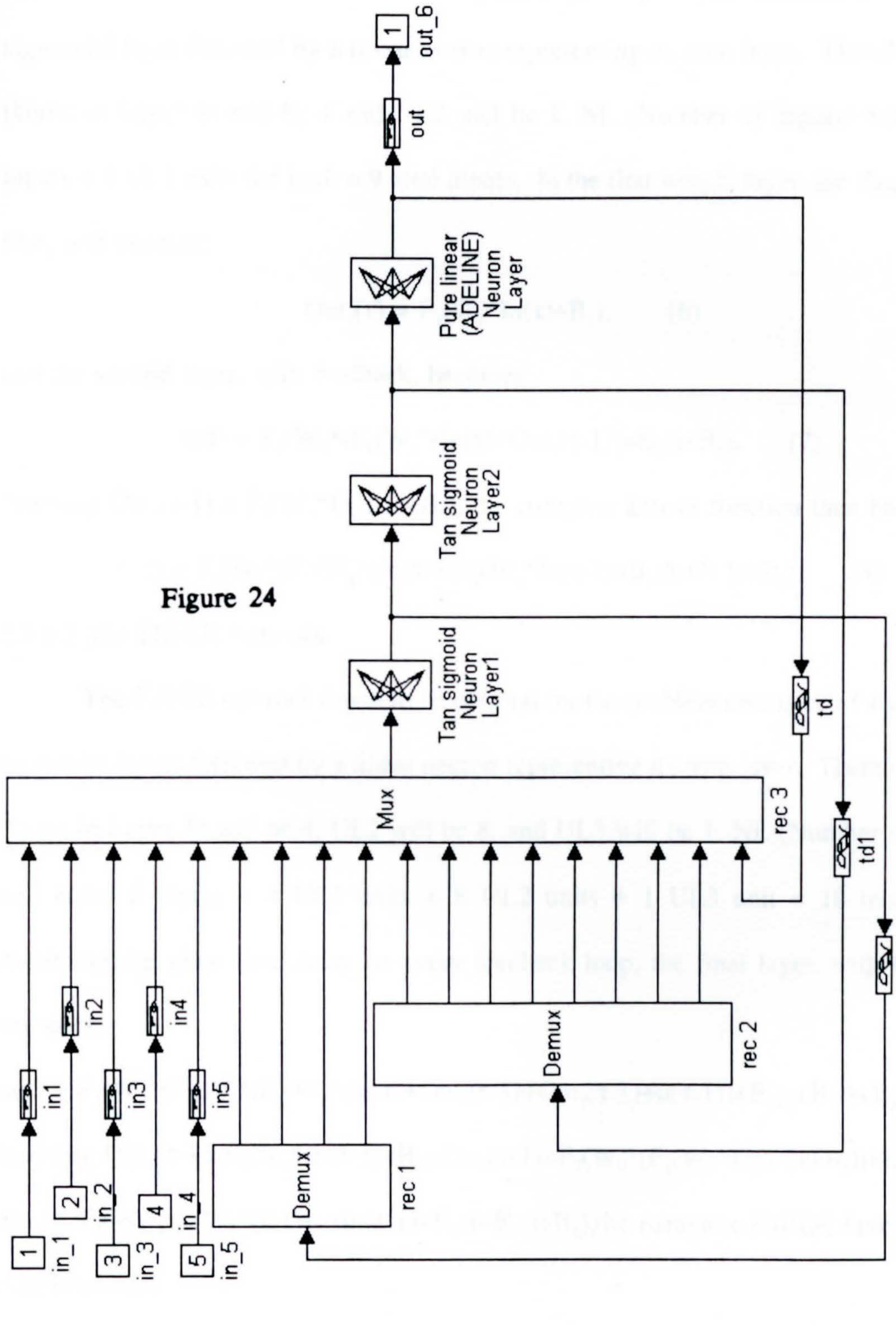


Figure 24

2.3.1.2 The Elman Network

The Elman network found to be optimal for the problem consisted of one 4 unit sigmoidal layer followed by a linear neuron representing its own layer. Therefore, UL1 (Units in Layer 1) will be 4 and UL2 will be 1. NL (Number of Inputs) will equal 5 inputs + 4 UL1 units fed back = 9 total inputs. In the first weight layer, the output vector Out_1 will become:

$$Out_1(\tau) = F_1(W_1*In(\tau)+B_1), \qquad (6)$$

and the second layer, with feedback, becomes

$$u(\tau) = F_2(W_2*(F_1(W_1*(In(\tau)+Out_1(\tau-1))+B_1))+B_2). \qquad (7)$$

Inserting $Out_1(\tau-1) = F_1(W_1*In(\tau-1)+B_1)$, the complete Elman function then becomes:

$$u(\tau) = F_2(W_2*(F_1(W_1*(In(\tau)+(F_1(W_1*In(\tau-1)+B_1)))+B_1))+B_2). \qquad (8)$$

2.3.1.3 The FSFER Network

The FSFER network found to be optimal for the problem consisted of 4 and 8 unit sigmoidal layers followed by a linear neuron representing its own layer. Therefore, UL1 (Units in Layer 1) will be 4, UL2 will be 8, and UL3 will be 1. NL (Number of Inputs) will equal 5 inputs + 4 UL1 units + 8 UL2 units + 1 UL3 unit = 18 total inputs. Assuming the same time delay on every feedback loop, the final layer, with feedback, becomes:

$$u(\tau) = F_3(W_3*(F_2(W_2*(F_1(W_1*(In(\tau)+Out_1(\tau-1)+Out_2(\tau-1)+u(\tau-1))+B_1))+B_2))+B_3). \qquad (9)$$

Inserting $Out_1(\tau-1)=F_1(W_1*In(\tau-1)+B_1)$,$Out_2(\tau-1)=F_2(W_2*(F_1(W_1*In(\tau-1)+B_1))+B_2)$, and $u(\tau-1)=F_3(W_3*(F_2(W_2*(F_1(W_1*In(\tau-1)+B_1))+B_2))+B_3)$,the complete FSFER function then becomes:

$$u(\tau) = F_3(W_3*(F_2(W_2*(F_1(W_1*(In(\tau)+(F_1(W_1*In(\tau-1)+B_1))+(F_2(W_2*(F_1(W_1* In(\tau-1)+B_1))+B_2))+(F_3(W_3*(F_2(W_2*(F_1(W_1* In(\tau-1)+B_1))+B_2))+B_3)))+B_1))+B_2))+B_3). \quad (10)$$

2.3.2 Neural Network Based Modelling

The more common feedforward networks (Figure 22), by their very nature, do not consider time delays. In a time dependent system, the network must be given time delayed signals explicitly at the input[4]. This, although effective, is a waste of valuable inputs and weights. Furthermore, it is a somewhat inflexible strategy in the sense that the optimum delay(s) must be selected by the designer, thus defeating the ability of the neural network to become a truly effective approximator. The simplified recurrent architecture represented by the Elman network in Figure 23 has the benefit of what may be considered to be a state feedback from the first layer to a second set of input weights. This feedback has a dual purpose: First, it permits better modeling of the internal dynamics of a complex system.[5] Secondly, it permits characterization of an indefinitely long time delay, or a combination of delays, different for various inputs or input characteristics. In effect, the feedback significantly enhances the ability of the neural network to model nonlinear dynamical systems.

The network of Figure 24 builds upon the basic recurrent by extending it to a full three layer architecture with feedback from each layer brought back to its own set of input weights. Also, each feedback incorporates independently tuned time delay elements which may be adjusted as small as the discretization step (the minimum for digitally simulated networks). This architecture differs from traditional recurrent nets in that typically, one would extend the recurrent by simply adding more sigmoidal layers to the Elman. This would imply that each layer has both inputs from the previous layer and a second set of inputs corresponding to its old outputs. This network was found to be more difficult to design and train than if all feedbacks were brought back as inputs to the very first sigmoidal layer as shown in Figure 24.

2.3.3 Determination of the Training Set and Algorithm

Present engine test techniques include a so-called *transient cycle*, driving the plant through a wide frequency range, and offering an excellent excitation signal for model development. All models are based upon a training set which consists of five input signals, and the actual system torque response, all sampled at 100 Hz. The inputs (1-5 on Figures 22-24) consist of present and past throttle commands, present and past speed commands, and finally, the difference between the past two throttle commands as seen below:

SPEED_CMD(t)

SPEED_CMD(t-1)

THROTTLE_CMD(t)

THROTTLE_CMD(t-1)

THROTTLE_CMD(t)-THROTTLE_CMD(t-1)

The penalty function for training (Γ) is computed as the sum squared error between actual system torque T and the networks computed torque shown as out_6 in Figures 22-24.

$$\Gamma = \sum_{i=1}^{N} \left(T_i - out_{6_i} \right) 2 \tag{11}$$

Weight updates are performed by the Levenberg-Marquardt[6] (LM) algorithm which is found to be significantly faster than traditional backpropagation. The LM algorithm is derived from Newton's method, and uses Equation 12 with J being the Jacobian error derivative matrix, μ being a tuning parameter and E the error vector to enact a weight change ΔW.

$$\Delta W = \left(J^T J + \mu\ I \right)^{-1} J^T e \tag{12}$$

The training process involved choosing the best of 5 complete training runs, where weights started at Nguyen-Widrow initialization,[6] and then iterated 30 times. It is found that, although the torque error levels off quickly by the completion of 30 iterations, it is of significant value to restart the network from scratch using a fresh set of randomly initialized weights. As often occurs with nonlinear problems, optimization algorithms (including those used for neural networks) have difficulty finding the global minima. While various techniques are built into both Backpropagation and LM to extract the global minimum from a highly nonlinear error surface, it is often helpful to initialize training at a variety of random positions in the error space, greatly increasing the chance that the global solution shall be located.

2.3.4 Model Training Comparison

Work was performed on a 486/100 microcomputer in the MATLAB/SIMULINK environment. Training time is a function of network size and architectural complexity. The simple Feedforward net, containing roughly 60 weights, takes approximately 30 minutes to train and only several seconds to execute a 100 second simulated segment. At the other end of the spectrum is the FSFER network with 112 weights and a complex feedback structure. This implementation takes approximately 150 minutes to train 5 full passes and 20 seconds to execute the trained network.

Qualitatively, Figures 25-27 show the torque output of the neural network superimposed upon that of the actual system for selected portions of the transient cycle. The time domain comparison reveals that both the Elman and FSFER architectures exhibit a fairly close approximation of the actual system, each with some level of steady state error and similar following of the transients (Figures 25-26). The Feedforward network, while still following the transients, appears very noisy, with a significant high frequency component riding on the intended signal (Figure 27). The accentuated high frequency components of the Feedforward network are equally apparent in the frequency response of Figure 29, whose reference signal appears in Figure 28. The Elman and FSFER networks show a somewhat closer result in Figures 30-31.

Quantitatively, the networks are each evaluated on the basis of the training set itself and two sections of the transient cycle for which they were not trained. While it is of interest to minimize error in all respects, it is the performance of the network (system model) in response to inputs which have not been previously experienced that is of critical value. To better characterize the issue, the magnitude response to both throttle

FSFER THROTTLE RESPONSE

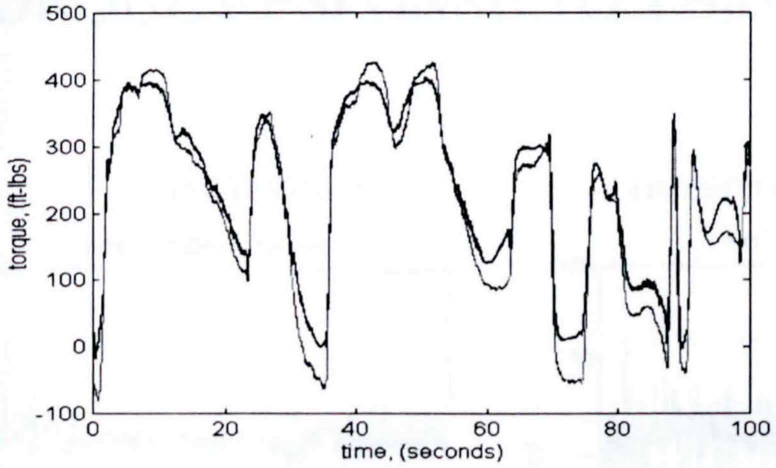


Figure 25

ELMAN THROTTLE RESPONSE

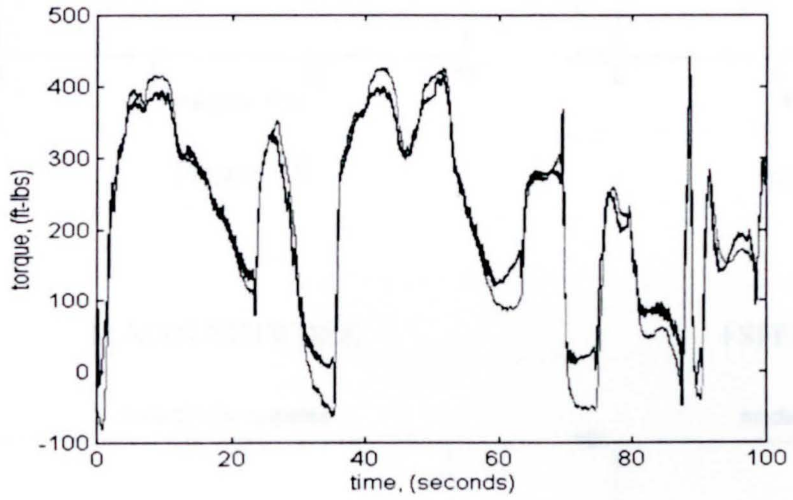


Figure 26

FEEDFORWARD NETWORK

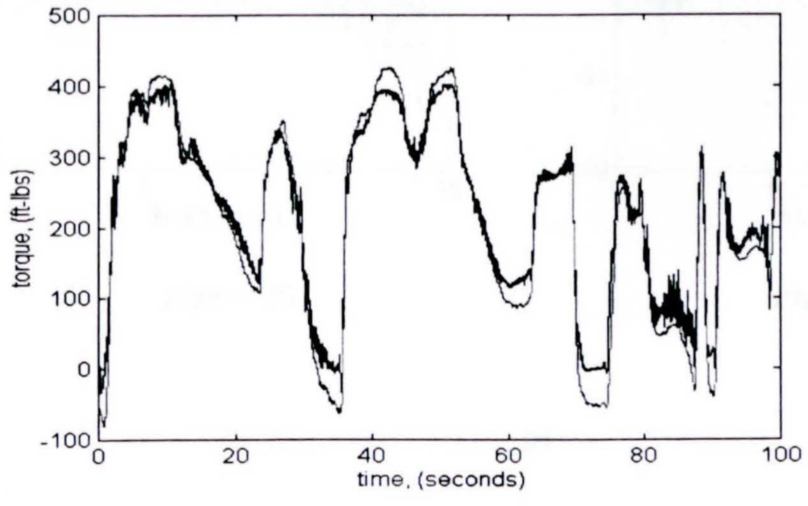


Figure 27

TORQUE RESPONSE TO THROTTLE INPUT

ENGINE TESTBED

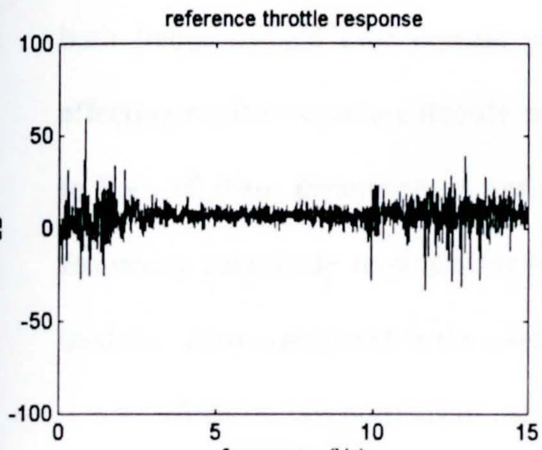


Figure 28

FEEDFORWARD NETWORK

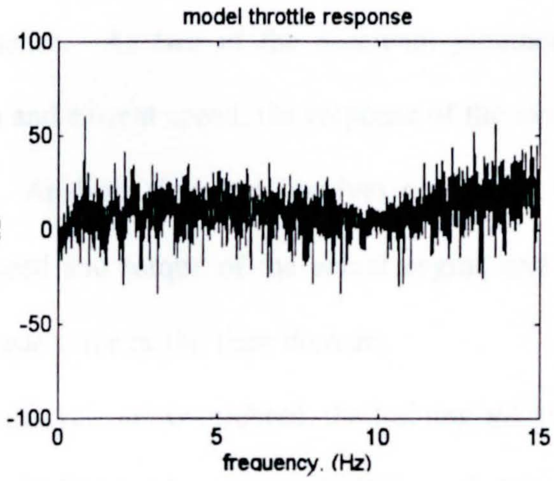


Figure 29

ELMAN NETWORK

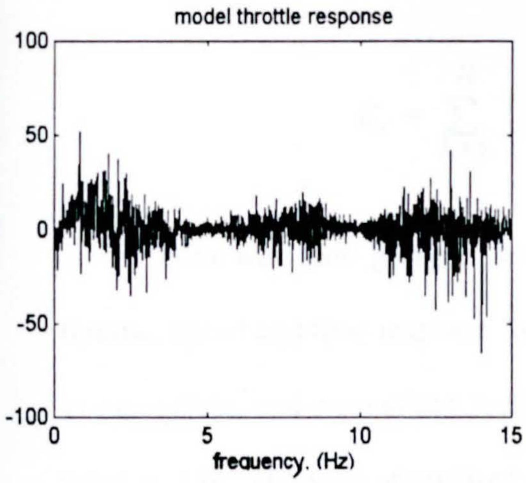


Figure 30

FSFER NETWORK

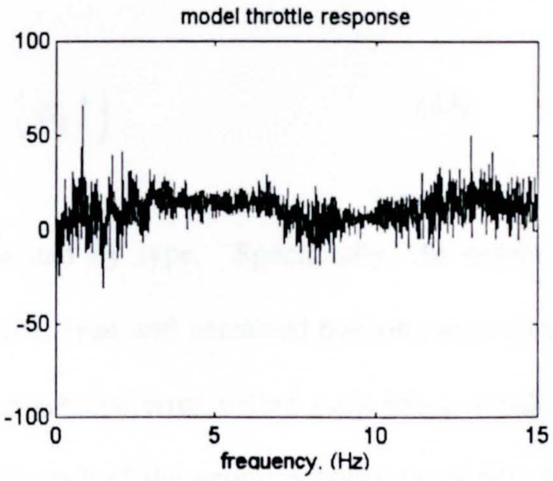


Figure 31

and speed inputs were computed and compared with the reference.[7]

The sum absolute error in the time and frequency response for the training set and two other data sets (which the neural networks had not been trained on) are then computed, and normalized to strip the units. Each network is evaluated on the basis of both frequency and time domain performance. As two of the dominant parameters affecting engine torque are throttle position and current speed, the response of the model to both of these parameters is evaluated. Analysis therefore involves comparing the frequency magnitude response for both speed and torque of the actual engine and all models. Also considered is the overall torque error in the time domain.

Three overall portions of the transient cycle are considered, the training set itself and two other portions for which the networks had not been trained. This evaluates the ability of the networks to model the actual dynamics of the problem as opposed to simply memorizing a trace. The total absolute error for each evaluation x is given by Equation 13 for N total points of reference R and model G.

$$E_x = \sum_{k=1}^{N} \left(\left| R_k \right| - \left| G_k \right| \right) \tag{13}$$

Data was then grouped and normalized by type. Specifically, the errors for throttle, speed and time response for each of trained and untrained portions were treated as equivalent, and normalized such that the greatest error within each group would be equal to 1.0. The total normalized errors for each of the neural network types was then totaled and represented on a scale of 0-1 for Figure 32, where it is seen that recurrent

QUANTATITIVE ERROR EVALUTION

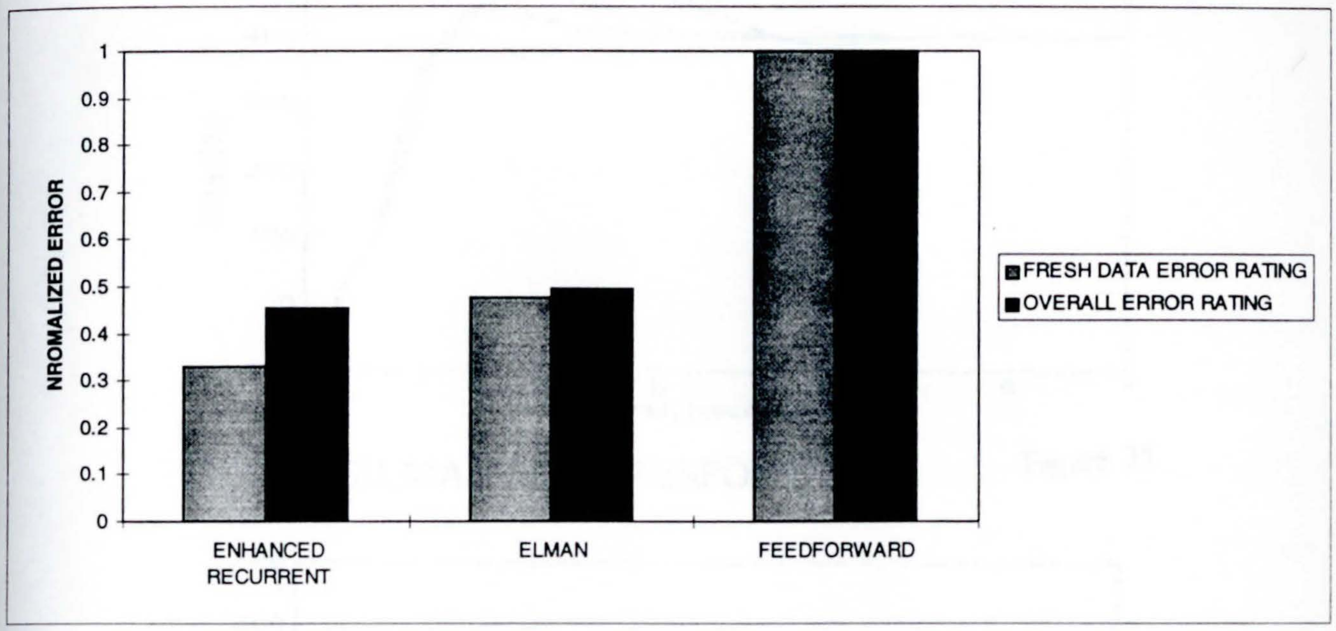


Figure 32

CONTROLLED SYSTEM STRUCTURE

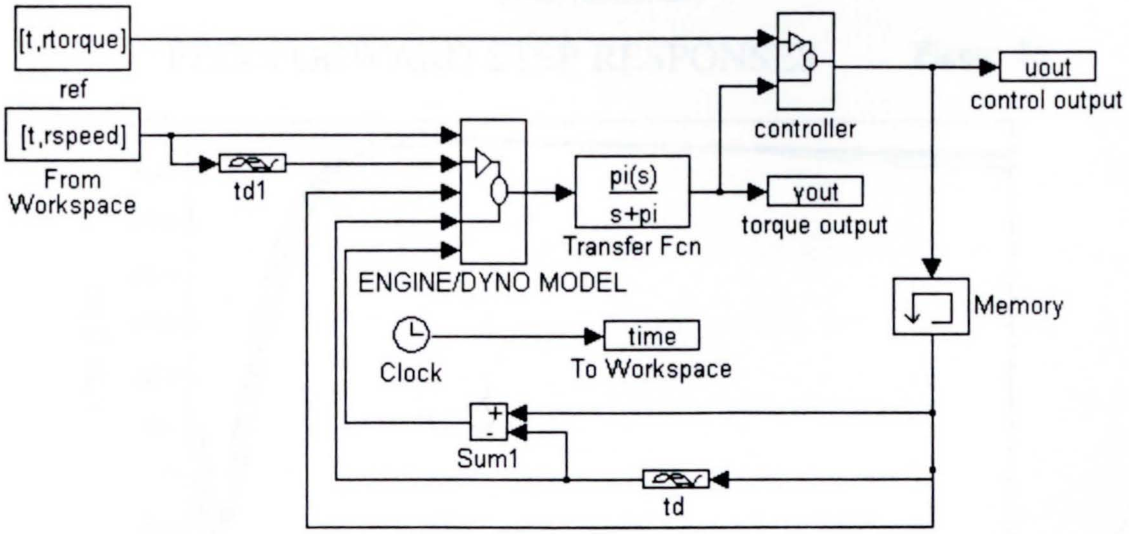
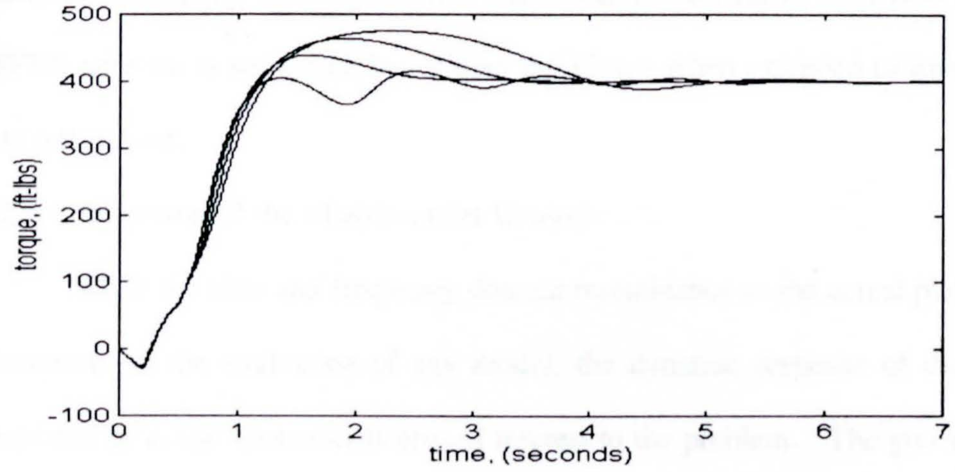
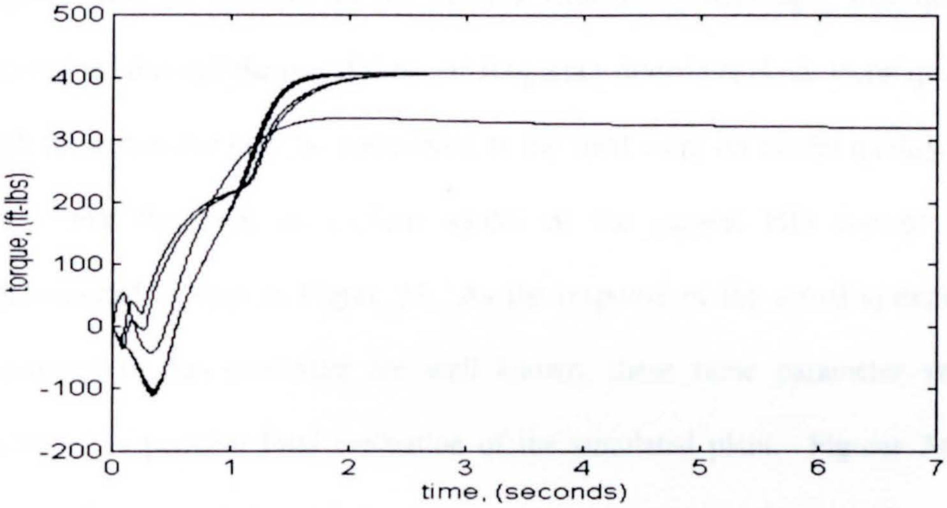


Figure 33

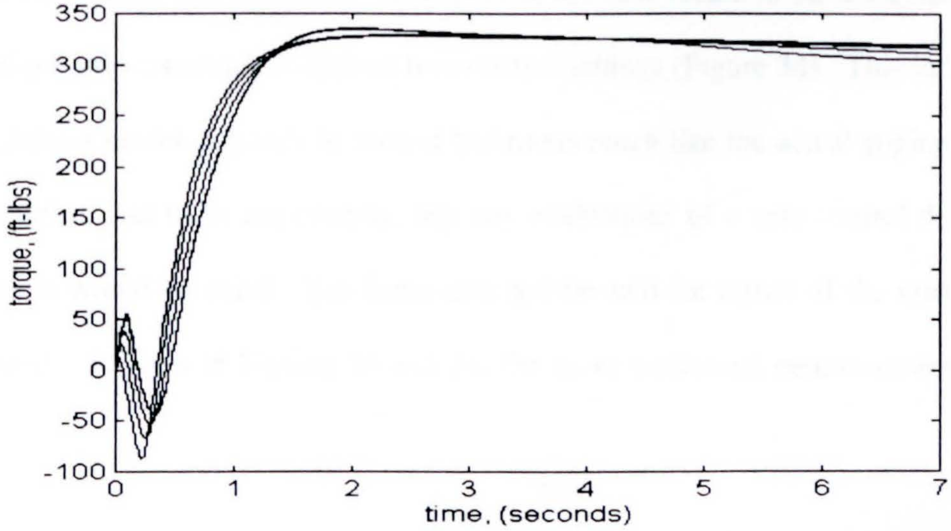
FSFER STEP RESPONSES **Figure 34**



ELMAN STEP RESPONSES **Figure 35**



FEEDFORWARD STEP RESPONSES **Figure 36**



networks are significantly better than traditional feedforward. It is also seen that the FSFER network is somewhat better than the Elman when extended to inputs for which it is not trained.

2.3.5 Comparison of the Models under Control

While the time and frequency domain resemblance to the actual plant are critical parameters in the evaluation of any model, the dynamic response of the model to a simulated or actual controller is also of interest to the problem. The given task implies that any resulting model must interact properly with the controller, and respond appropriately to changes in the control structure. Although, this quality may be ascertained through the use of time and frequency domain analysis techniques, evaluations with the controller may be considered as the final word on model quality.

For this test, an explicit model of the current PID control structure was implemented as seen in Figure 33. As the response of the actual system to parameter variations in this controller are well known, these same parameter variations were recreated to perform final evaluation of the simulated plant. Figures 34-36 show the results of proportional and derivative parameter variations in the controller, and the plant's corresponding step response. As expected, the new recurrent network has a completely unique step response for each of four control settings (Figure 34). This indicates that the recurrent model responds to control variations much like the actual engine testbed being modeled, and more importantly, that any evaluations of a new control design with this model would be valid. The same may not be said for either of the other two models tested. As seen in Figures 35 and 36, the more traditional neural network models are quite insensitive to control variations, indicating that any evaluation of a unique control structure with these plant representations would be highly uninformative.

2.4 Future Directions for Modelling with FSFER

Current efforts are directed toward both control development on the present model and model refinement in terms of both architecture and training. The concept of a hybrid network incorporating both recurrent and feedforward elements (separably trained) shows significant promise. In terms of control application, it is believed that both the performance and robustness of the control system can be significantly improved over that offered by the existing proportional derivative strategy. Also, further applications of the enhanced recurrent network are being investigated including the modeling of full vehicle dynamics for the development of automatic drivers, and the extension of the new architecture for use as a feedback controller.

It is found that the enhanced recurrent architecture described is ideal for modeling nonlinear dynamic systems. This architecture has been developed and trained for the heavy duty engine testbed application of present interest and compared favorably over other neural architectures. Future work involves development of a unique control strategy for the engine testbed application as well as application of the FSFER architecture to other modeling and control problems.

2.5 Incorporation of FSFER into a Hierarchy

A hierarchical control composed of fuzzy logic, neural networks, and expert systems will demonstrate that it can not only execute a realistic task in a crowded environment, but can learn to accomplish similar tasks in a more efficient fashion at a later time. Real world complex systems are characterized by poor models of an uncertain environment, high dimensionality and nonlinearity of the decision space, hierarchies of influence/complex information patterns, multiple performance criteria, distributed sensors and actuators, and high noise levels, all resulting in computational complexity for control schemes. Biological systems have shown that they can best handle complexity, uncertainty, redundancy, noise rejection, fault tolerance, and signal fusion.

The possible application areas are many. Current incarnations of vacuum cleaning and lawn mowing robots must shut down for unforeseen circumstances and wait for a human to return. Assembly robots, taught teach points for processes such as welding, can run day and night, until their components fail. With a method of diagnosis tied into their control, maintenance and programmers can be warned of an impending line down condition while they are still on duty. Welding robots for joining variably curved shiny surfaces would have a lower reject rate if they could learn from visual feedback that they were currently making a bad weld, and correct it before the seam is completed. Another case of a constantly drifting system is when a robot must load semiconductor wafers on an evaporator dome. Warpage from constant reheating causes each mounting depression on every wafer dome to be differ, setting the stage for wafer mismounting. To complicate the issue, visual confirmation of a good mount is confused by reflections from the polished stainless steel equipment in all directions. Even human driver teleoperators manipulating unstable loads can benefit from learning. A controller is needed that can sense the *intentions* of a human that is overreacting to a potentially unstable grasp, based on experience. For quadriplegics and other wheelchair disabled, a learning adaptive controller can mean the difference between constant supervision and relative independence. A mouthpiece or head mounted laser indicator controller, in concert with to a learning robot attached to a wheelchair, should be able to adapt to everyday minor calamities and emergency situations instead of having to wait for service personnel to arrive.

2.6 Proposed Hierarchy for This Work

Given that online adaptive control is a difficult topic for neural networks, the goal of this effort is to demonstrate some positive results with the chosen combination and modification of approaches, given a weight training method that is timely for real world applications. The hierarchy developed (Figure 37) is intended to fall back to a stable control in a new situation, then learn how to deal with the new difficulty. From the standpoint of cooperating arms or redundant joints, the hierarchy will provide a choice of stable controllers to choose from, a smooth stable transition from one controller to the next, and a more optimal solution for the next time a similar set of operating conditions is encountered.

2.6.1 Most Local Level: Diagnostic DC Motor Control

A diagnostic adaptive DC neural motor controller is developed to compensate for parameter variation at the joints, and then pass information it gains on to higher level control and planning. The goal is minimization of human intervention due to mechanical

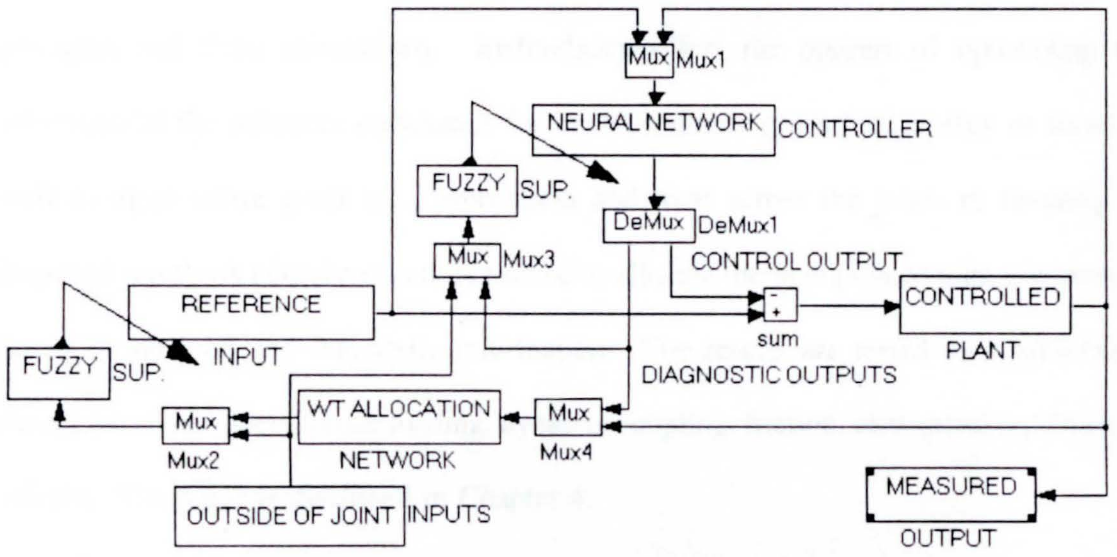

Mux
Mux1
NEURAL NETWORK
CONTROLLER
FUZZY
SUP.
DeMux
DeMux1
Mux
Mux3
CONTROL OUTPUT
REFERENCE
INPUT
sum
CONTROLLED
PLANT
FUZZY
SUP.
DIAGNOSTIC OUTPUTS
Mux
Mux2
WT ALLOCATION
NETWORK
Mux
Mux4
OUTSIDE OF JOINT
INPUTS
MEASURED
OUTPUT

Figure 37

wear and unforeseen degradation in the performance of one joint so that redundant joints can backfill until maintenance is convenient. Consistency, safety, and stability are important considerations. Some comparison is made between using time domain versus frequency domain information in doing diagnostics (i.e., helicopter gearbox). This topic is discussed in Chapter 3.

2.6.2 Coordination of Many Joints: Redundant Axis Robot Control

With only six joints there is one way for the end effector to achieve three positions and three orientations. Redundancy offers the options of optimizing the allocation of the reference commands for obvious simple goals (i.e., energy or time) as well as more subtle goals (i.e., even stress and wear across the joints or favoring an impaired joint). A neural network is trained to allocate the manipulator joint commands, based on position and diagnostic information. The results are tested on a simulation model which considers torque loading, dynamic coupling, friction, centripetal and Coriolis effects. This topic is discussed in Chapter 4.

2.6.3 Working Toward a Goal: Grasp Planning with Tip and Slip Control

A neural object approach vector classifier is also developed for choosing the best first grasp for a class of objects. An initial determination of approach vector is made via stability rules. As an object lift is attempted, fuzzy logic is used to correct for tip and slip. Then an object grasp rank based on the success of the fuzzy attempts is used as feedback for the neural object category network. Objects are clustered based on trial grasps, not by human perception. A simulation of the fuzzy grasp tipping algorithm is demonstrated, as well as the ability of a neural network to replace fuzzy controller across a wider range of variable parameters. This topic is discussed in Chapter 5.

2.6.4 Working with Other Robots: Multiple Arm and Object Planning

The last topic, parallel task planning, utilizes a self-organizing neural network to extract an appropriate selection of plans from a stochastic petri net. A block manipulation and stacking task using two robot hands is demonstrated in simulation. This topic is discussed in Chapter 6.

2.7 Extensibility of the Hierarchy

A similar hierarchy is developed for an detecting and driving automobiles on a chassis dynamometer in Chapter 7, where additional processing required to interpret the sensory information. Successful at recognizing the frequency signatures of vehicles it has never been trained on, as well as demonstrating promising modelling and control ability, this hierarchy lends support to the extensibility of the concept to many practical applications.

References

[1]Tascillo, M., Tascillo, A., and D. Gong."Neural Network Based System Identification of an Engine Testbed," **Proceedings IPC '95 Workshop**, Detroit, Michigan, May 9-11, 1995.

[2]Winston, Patrick Henry, **Artificial Intelligence**. New York: Addison-Wesley, 1992.

[3]Kawarabayashi, S. and T. Fugii: "Design of Optimal Servo-Systems for Engine Test Bed by ILQ Method", **Proceedings of 28th Conference on Decision and Control**, December 1990.

[4]Rumelhart, D.E., G.E. Hinton and R.J. Williams: "Learning Internal Representations by

Error Propagation." **Parallel Distributed Processing**, vol. 1, D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., Cambridge, MA: MIT Press, 1986.

[5]Caudill, M. and C. Butler: **Understanding Neural Networks, Volume 2: Advanced Networks**. The MIT Press, Cambridge, MA, 1992

[6]Demuth, H. and M. Beale: **MATLAB Neural Network Toolbox.** The Mathworks, 1994.

[7]Proakis, G. and D. Manolakis: **Introduction to Digital Signal Processing**. Macmillan, NY, 1988.

CHAPTER 3:

DIAGNOSTIC DC MOTOR CONTROL

3.1 Neural Networks that Diagnose

Although recent literature indicates a proliferation of neural network applications, a significant challenge remains in the area of control engineering. A typical parallel algorithm[1] can effectively execute one repetitive, short term robotic task involving the estimation of parameters, command of motor joints, planning of trajectories, and grasping of simple objects. In an increasing number of applications, however, product runs change and interchange rapidly requiring frequent adjustment, retooling, and maintenance. It would be desirable to implement an algorithm that could reliably adapt to mechanical wear and product variation, thereby increasing production time between instances of human intervention.

Employment of neural networks to learn the entire robot control task from sensor to actuator or end effector has been successful for two or three degree of freedom systems,[2,3] but attempts to apply these to higher order, more realistic robots have been discouraging.[4] The most common assumption of conventional parameter estimation techniques is that initial conditions be close to their target values. One way around this limitation is to employ a robust control scheme that performs acceptably across all expected parameter values, but even in this case a partially known linear time-invariant system is assumed.[2] This approach may be inadequate when the control requirements extend simple, usually planar demonstrators to three dimensional robots with redundant joints performing high speed, precise manipulations of unstable, variable loads in cluttered environments. Neural networks can extend the capabilities of the controllers they were modelled after,[6,7] but once the task becomes too nonlinear, outputs must be supervised (i.e., with an expert system[8]) to correct for steady state errors and instabilities.

A diagnostic output or node can be defined as a neural network output whose purpose is to infer a nonlinear system state or intended reaction condition, to be used or interpreted by some other algorithm. Some research has begun recently with diagnostic outputs. Abu-Mustafa refers to "hints" in his work,[9] where extra outputs are added for the sole purpose of aiding the neural network in its internal representation of the nonlinear relationship intended for the first output. The hints are extra information the programmer happens to collect for each set of training data which, by contributing to the hidden layer weight updates, bias the network's approach to learning the original relationship. From another point of view,[10] helicopter gearbox frequencies are monitored by a neural network with an initial Fourier transform layer. All outputs are diagnostic, and their values are trained as failure present or not present -- the failure output with the highest value wins. From a third perspective, model reference control, neural networks have been successfully used to model the controlled plant so that the standard model reference control gains can be determined adaptively.[11] Hints have also been interpreted by an expert system when a neural network can not resolve steady state error at values near zero.

The next logical step is to employ the diagnostic nodes of a model or controller neural network for better selection of neural network weights. It is also logical that these diagnostics can also be used for higher level planning, such as for the redundant joint allocation of Chapter 4. The author's contribution[12-14] is the extension of diagnostic outputs as direct contributors to modification of the controller's own weights, as well as inputs to other levels of the hierarchy. A successful method of stably switching from one set of controller weights to another is demonstrated and a family of stably transitioned neural networks are trained to model a range of individually tuned PID or fuzzy controllers across the anticipated diagnosable workspace.

3.2 Diagnostics for Drifting Systems

A neural network controller is therefore designed that will first acceptably control a plant whose parameters drift with time, then transition to a specifically trained *child* controller based upon a stable *parent* neural network if increased accuracy is necessary and available. The controlling neural network itself possesses efficiently trained outputs to diagnose drift of crucial plant parameters. For this approach results are demonstrated for two plants, one linear, the other nonlinear. A demonstration of diagnostic outputs on a model neural network will be shown in Chapter 7.

3.3 Why Diagnose at the Motor Level

When faced with predicting the behavior of such a highly complex and nonlinear control task, a human will split the problem up into manageable tasks. With this approach in mind, some neural network based controllers[15,16] have broken up the dynamic equations for a specific robot into small functional blocks, and then developed optimal linear combinations of these blocks by training. Unfortunately, robot configuration flexibility and some nonlinear effects cannot be properly addressed. Attempts to accommodate these nonlinearities on-line by adjusting more than one layer of weights via a supervisory context network may lead to instability.[17] A similar approach was attempted by Yamaguchi et.al., [18] where a fuzzy controller blended two neural network gain schedules. It performed no diagnostics, but used fuzzy membership functions to ascertain a continuous altitude-dependent combination of neural controller outputs appropriate for control. The initial control, robustly stabilizable for all conditions, was used first as the default, but as learning progressed, it assumed an overlapping role between two more specialized networks, one of which exhibited robustness for high disturbance-low altitude flight, while the second was tuned to higher altitude flight. The neural architecture employed was Kosko's bidirectional associative memory (BAM), which memorizes input/output pairs via the reverberation of corrections using a correlation matrix and its transpose. Extrapolation to a large number of flight schedules was not discussed.

A control network is desired that is both sensitive and adaptive to the parameter variations of an arbitrary robot such that prespecified performance and robustness criteria shall be met. Inverse kinematics and dynamics can be estimated for a given robot in many ways, but the effect of parameter variation is most noticeable at individual joints. The function of a robot joint is theoretically simple: at any given moment, it must rotate clockwise or counterclockwise a specified distance. The movement control problem can be complicated, however, by variable parameters such as initial position with respect to gravity, load and environmental disturbances, sensor noise, friction, and beam stiffness. Any combination of the above can compound errors across many joints and result in excessive overshoot and/or settling time.

3.4 What to do with a Diagnosis

It was observed in previous work[19] that a standard feedforward backpropagation neural network had little trouble learning the three dimensional inverse kinematics relationship for a five axis robot, but fine tuning of the position and orientation across the entire workspace was a more difficult task. A solution[20] to the accuracy problem has been attempted by training networks to operate within a sequence of small windows spanning the workspace. However, sudden transitions between window dedicated networks can result in unacceptable robot operation.

Various researchers have attempted to add intelligent decision making to alleviate performance degradation caused by these transitions. Jordan and Jacobs[21], for instance, utilize a supervisory gating network which determines the control signal based on the weighted contributions of specialized expert controllers. Pomerleau's ALVINN[22], which interprets camera images to direct an automobile, employs a supervisory algorithm which chooses a steering command by gauging the importance of the suggestions of individual expert networks specializing in highway driving, off road driving, avoiding obstacles, etc. Upon implementation, data input interpretation difficulties have persisted, periodically confusing the networks. It is difficult to teach an algorithm a continuous sense of the entire driving experience when the basic driving task is interrupted by somewhat cyclic environmental phenomena including turns, obstacles, traffic patterns, and variably relevant changes in road surface, scenery, and weather conditions, whether multilayer perceptrons[22] or ART2 networks with edge detection preprocessing[23] are employed. These environmental factors must somehow be incorporated into the training set and accounted for by the neural network chosen. Voting among better options[24] can be an improvement over a single choice of best network -- although the highway network may be mostly correct that remaining in its lane is optimal, a small tree must still be avoided via the input of an obstacle avoidance network (which should contribute a large weighted vote beyond a given threshold).

It is known that accuracy of a neural net based controller is affected by the way a training set is generated. It has been suggested[19] that a training set accumulated along a spiral shaped trajectory of points within a small volume can considerably increase the accuracy of full size test trajectories spanning the workspace. Once the network extracted the inverse kinematics, the relationship was valid everywhere. An approach is desired that incorporates as much knowledge about an unknown dynamic system as possible into coordinated, manageable, and stable subcontrollers which are not restricted to a small neighborhood of applicability and do not require a pre-rehearsed sequence of snapshots from initial conditions to target.

3.5 Problem Approach

The goal of the proposed neural network based controller is to generate a control effort for the system of interest, minimizing some performance criterion. This problem can become complex due to non-stationary effects such as drifting plant parameters. These issues are addressed by system adaptation, which results in building an additional control loop responsible for controller characteristics. A neural network based approach has the potential for incorporating both the control and adaptation tasks in the same control procedure, providing that a rational training set has been generated. Assuming no direct state or other parameter feedback, only integral and derivative variations of system output error and control effort are used as inputs for the networks. Neural network diagnostic nodes could have been used to simply adjust a traditional controller's gains, but the pole placement controllers used as ideal targets for each child network must assume full access to all system states, and they often generate unrealistically high feedback gains in order to achieve an ideal response (i.e., minimal settling time).

While the control problem is the primary task of the neural network, some valuable diagnostic information can be generated. The advantage of adaptation emerges when additional diagnostic outputs can recognize the beginning of a parameter shift, revert to a network that can handle all anticipated cases to an acceptable degree, then execute a smooth transition to a chosen specialized, more accurate network. Sharper resolution between parameter values are obtained when the diagnostic values used for inertia values of

$$\{.003, .0045, .01, .02, .03, .04, .05, .06, .1, .5\}$$

are converted to targets of

$$\{-4\ \ -3\ \ -2\ \ -1\ \ .1\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\}$$

for training, allowing for easier interpretation of network results when the practical values are very close together.

In order to accomplish these goals for a given system, a parent network is trained to thoroughly learn the desired control functional relationship across all anticipated operating conditions. This stable control can then be enhanced by initializing specialized child networks with the parent's weight values and exposing the network to a new training set devoted to a system with a varied parameter. Having achieved an already similar relationship, the child will not need to radically alter its weights, offering a more continuously stable transition from the parent system. A separate diagnostic unit for each variable parameter is added to the parent network by freezing all existing weights of the parent network and training a new set of linear connections leading directly to the diagnostic unit.

To demonstrate the approach two types of systems are considered, the first aimed at the evaluation of the efficiency of the adaptive control procedure. A simple linear controlled plant,

$$\frac{1}{(s+p)}, \tag{1}$$

is designed with a feedback controller,

$$10 - p, \tag{2}$$

that is guaranteed to provide a first order response with a 0.4 second settling time. A neural network was trained to both imitate this adaptive controller and identify any drift of the time dependent parameter p, potentially a value between 1 and 5. It was found that linear connections from the network inputs to the diagnostic output were all that was necessary for an accurate diagnosis of p, and a linear adjustment from one set of weights to another offered a smooth transition in control.

The simulation diagram used to train a neural controller for the second system, a DC motor with load bearing arm, is shown in Figure 1. A pole placement state feedback controller is first found for each set of variable plant parameters, in this example load inertia j_{load}, load friction b_{load}, and joint arm twist stiffness, k_{load}. Secondly, the network is trained with an enhanced backprop-based algorithm that has been designed to repeatably converge within a minimal number of epochs (Figure 2) and provide a stable and somewhat accurate control command, u_{new}, for all example parameter variations.

Control commands for references of 1 and 2 for a plant with load inertias of .03 and .06 are shown in Figures 3 and 4.

A weighted error evaluation function is also used for training this output, varying the emphasis on initial transient control efforts versus those exerted near steady state. For the DC motor, linear connections from both the network inputs and the second layer of sigmoidal units were necessary for proper identification of drifted parameters.

The architecture of the neural motor controller network is shown in Figure 5, where the hidden layer units are hyperbolic tangent sigmoid activation functions and the output layer is linear. Both neural network inputs and second hidden unit layer neurons contribute weighted connections to the output nodes. The inputs to the motor network are the error between the reference and the plant output, the error doubly integrated, the previous control input to the plant, and its derivative and second derivative values.

3.6 Diagnosis Results

Figure 6 shows the first order pole placement target response superimposed with both the slower, highly overshooting, uncontrolled plant and the improved, realistically implementable neural response. Figures 7 and 8 illustrate a series of linear system and motor neural controller responses to commands for a given set of parameters superimposed with the reference. Figures 9 and 10 show filtered responses to drifting parameters, closely fitting the actual values, some of which vary considerably from the first "normal" parameter. The actual output of the node follows a shape similar to the control output u_{new}, but the magnitude information can be easily and accurately extracted.

Noise was then added to all inputs, resulting in the parameter diagnoses of Figures 11 and 12. Finally, parameter shifts were imposed upon the neural controlled system during implementation, an example of which is shown in Figure 13. Exhibiting a slightly different transient response, the new system picks up smoothly at a reference of two where the parent left off at the reference of one.

3.7 An Illustrative Example

A series of figures will demonstrate the control procedure. A normal plant, with a load inertia of .03, is commanded to move from 0 radians to 1 and then 2 radians. The parent network controller, optimal near the normal operating parameters of load inertia j_{load} =.03, load friction b_{load} = .38, and beam twist stiffness k_{load} = $5x10^4$, will accurately execute this command (Figure 14). Interpreted by a fuzzy supervisor as having remained within acceptable limits for the current controller, the parent controller remains active.

After the motor arm is returned to a position of zero radians, an additional external load is added, doubling the previous value to .06. A command is then given to move to one radian, resulting in an acceptable, stable response with some steady state error. While this command is executed, the diagnostic output for load inertia interprets the new plant as requiring a specialized child network, noting that the averaged value of .2693, formerly closest to the normal .1 target for j_{load} of .03, has now shifted to 3.0916, closest to the target of 3 for a j_{load} of .06. The weights are linearly drifted to the child values as the plant is commanded to move to two radians, demonstrating a smooth transition in control across all sets of weights to the desired steady state value. Figure 15 compares the

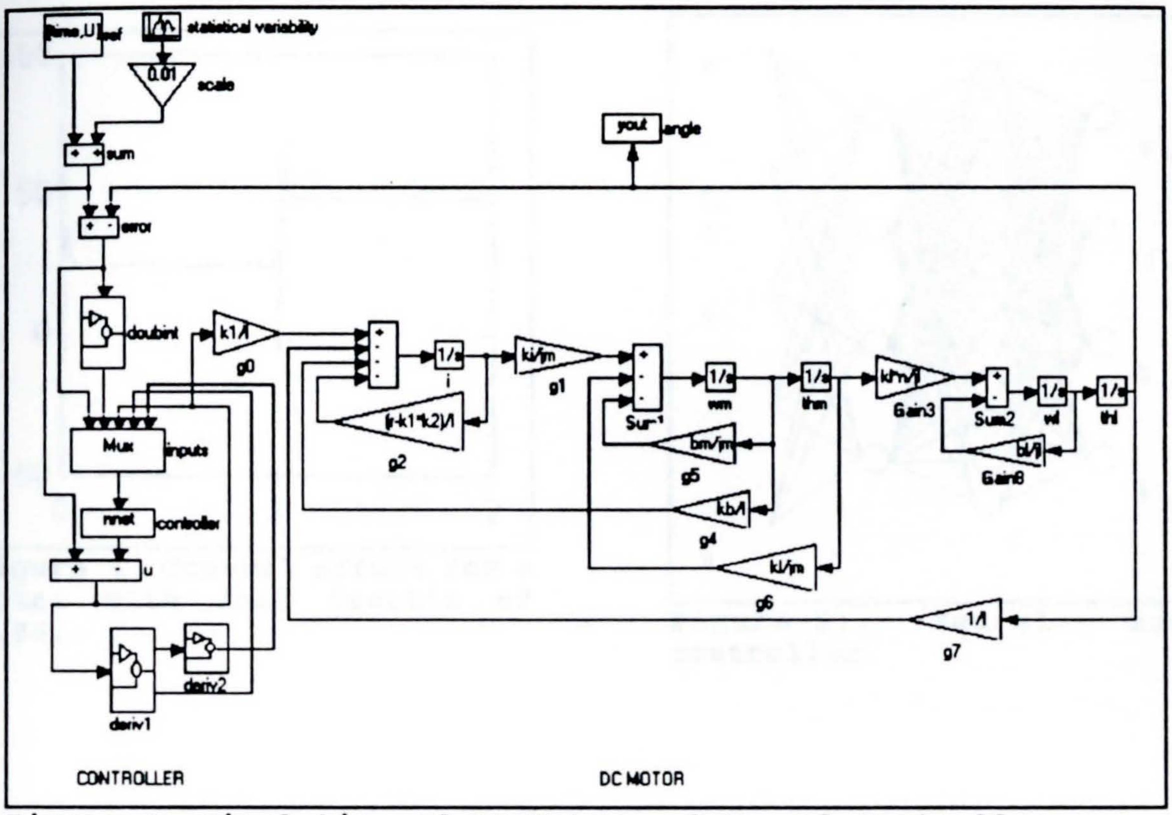


Figure 1: Simulation of DC Motor and Neural Controller.

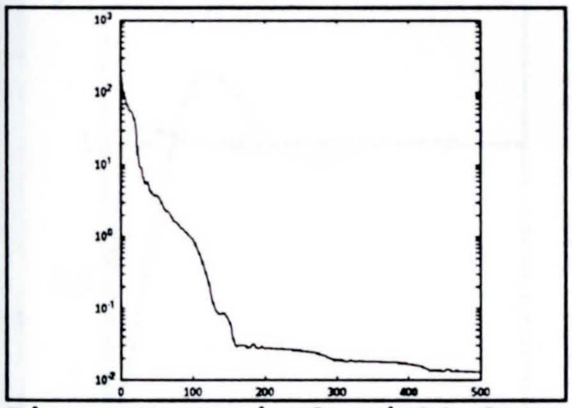


Figure 2: Typical weighted sum squared error curve.

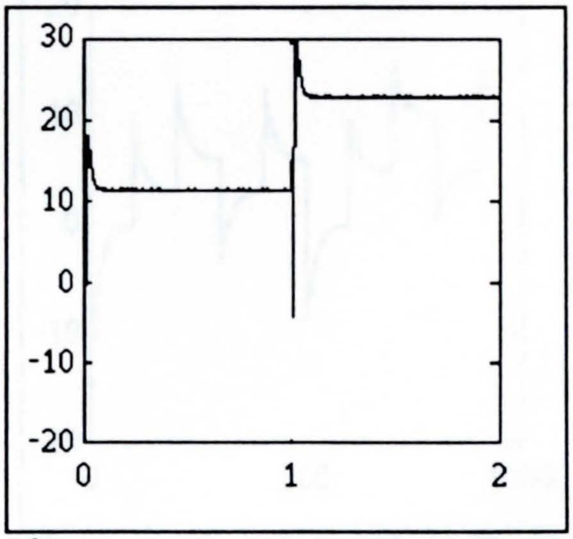


Figure 3: Control effort for a motor with a load inertia of 0.03.

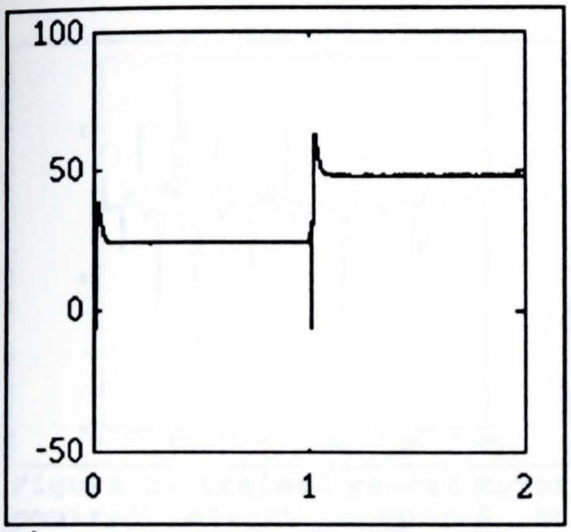


Figure 4: Control effort for a motor with load inertia of 0.06.

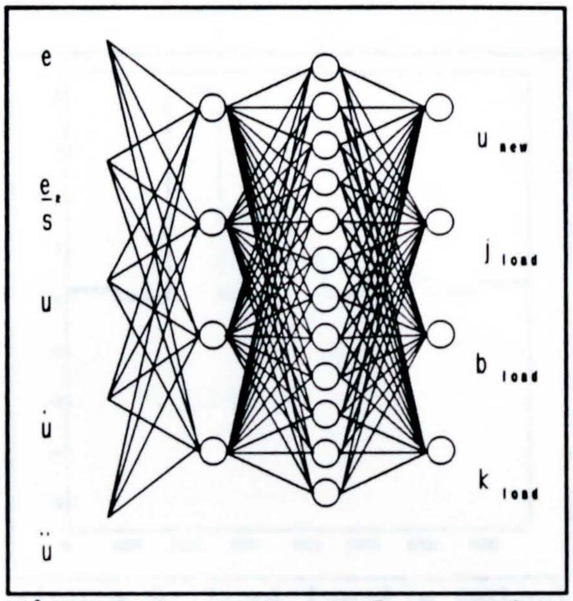


Figure 5: Neural motor controller.

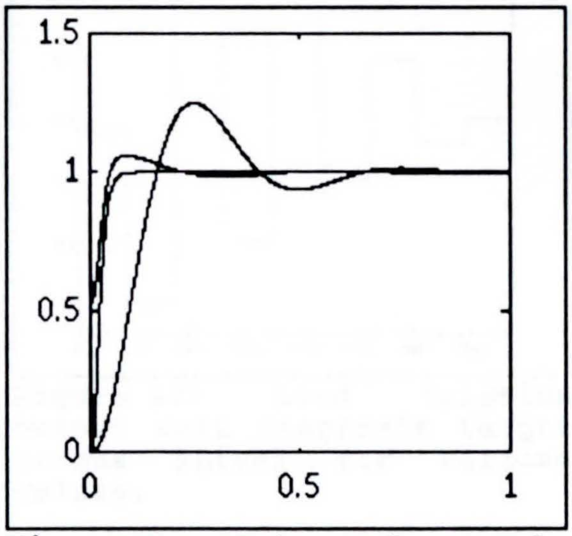


Figure 6: 1st order pole placement, high overshoot uncontrolled plant, & neural controlled.

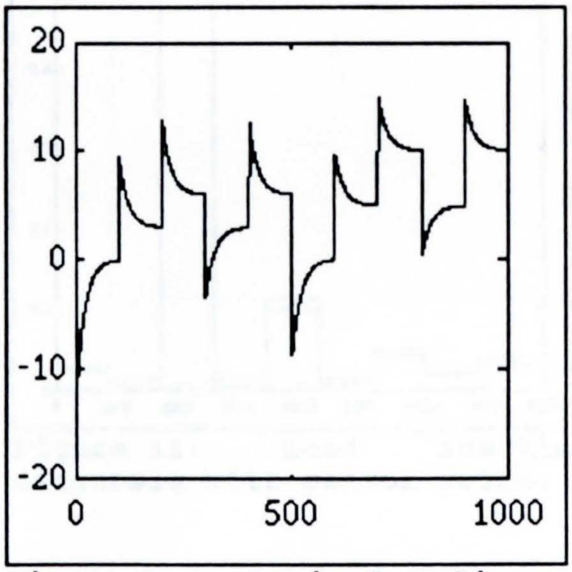


Figure 7: Trained linear system neural control efforts for small and large parameter p given a series of reference commands.

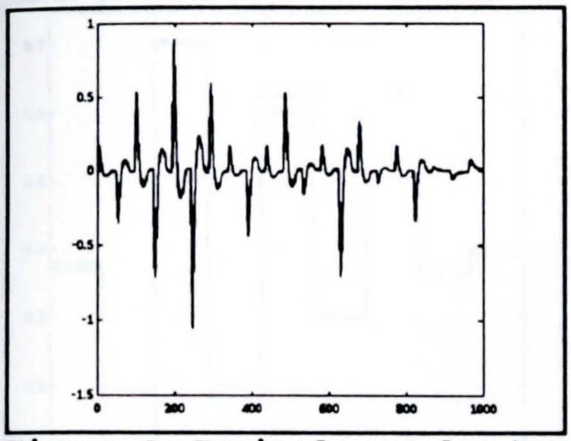


Figure 8: Trained neural motor control effort compared to pole placement effort for large and small inertia values.

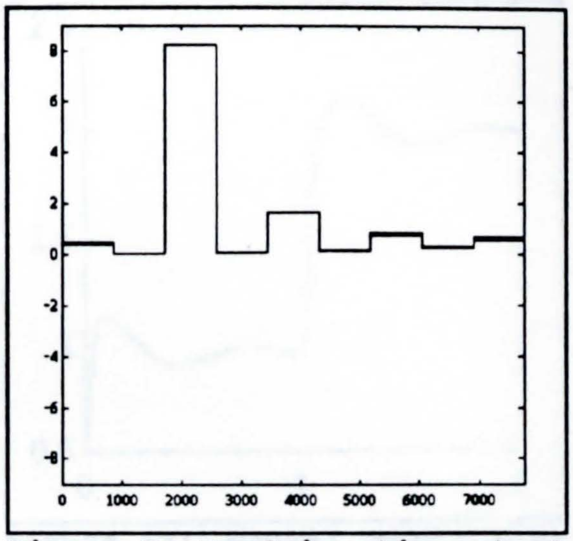


Figure 9: Load inertia output unit diagnosis, target versus actual for various values, as average across 1000 samples.

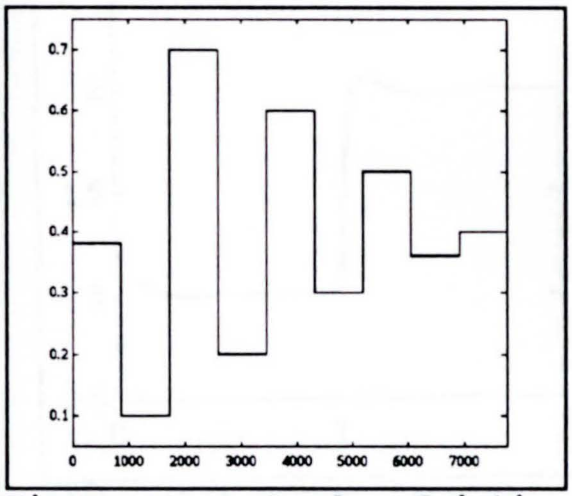


Figure 10: Load friction output unit diagnosis target versus actual for various values.

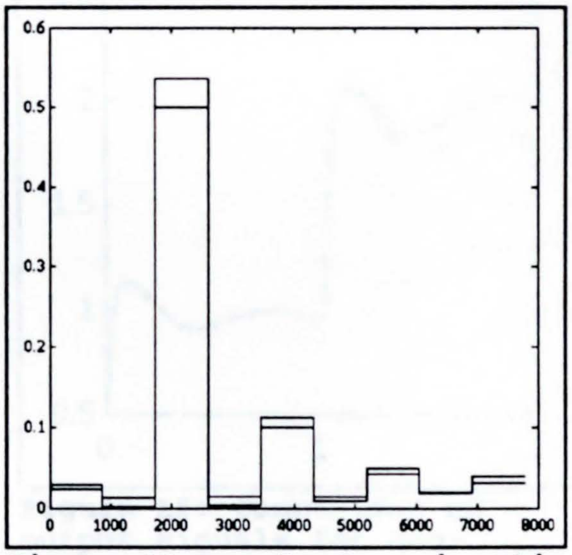


Figure 11: Load inertia diagnosis with sensor noise.

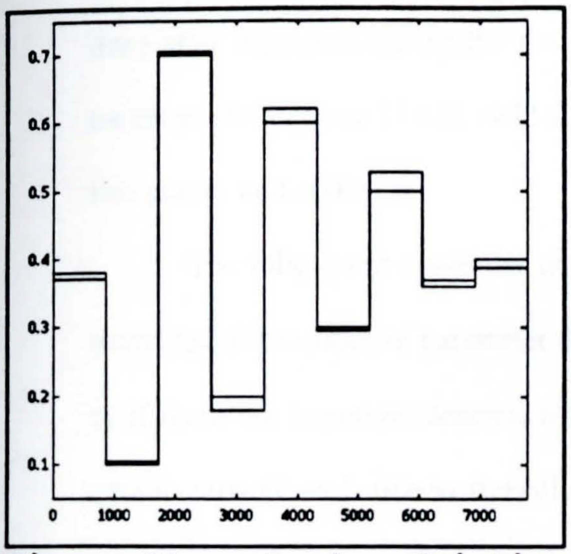


Figure 12: Load friction diagnosis with sensor noise.

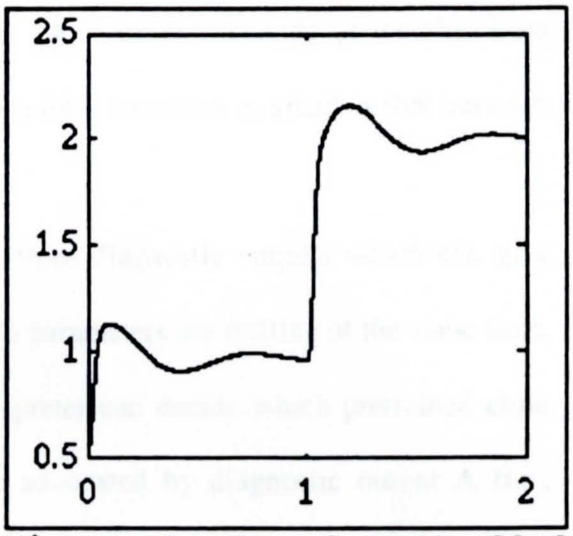


Figure 13: Neural controlled DC motor system response to a reference and parameter change.

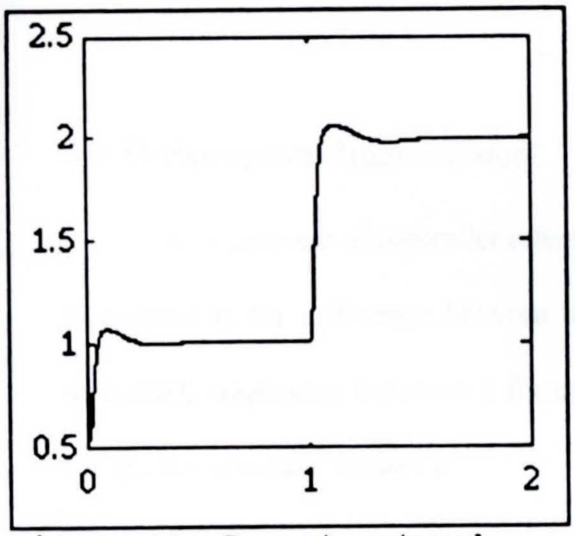


Figure 14: Parent network response for the normal plant.

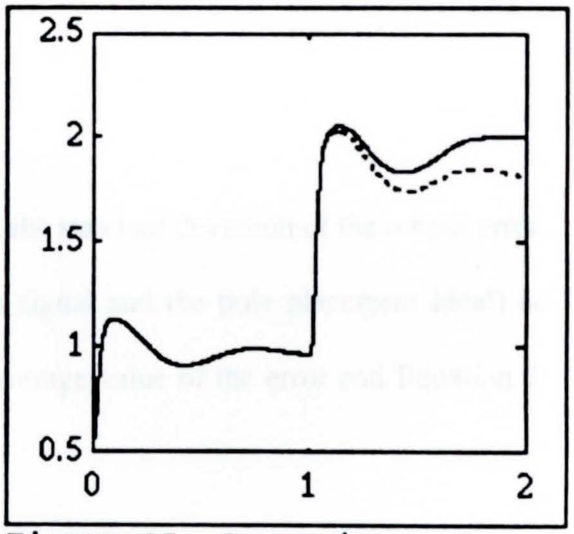


Figure 15: Comparison of output signals for non-adaptive parent (dotted line) and parent with drift adaptation to child controller (solid line).

response of the parent controller with no drift (dotted line), to the drifted response (solid line) after diagnosis during the first command. An instant exchange of weights from parent to child Figure 11 will yield similar results for a transition as small as that between this parent and child.

The robot joint controller network has three diagnostic outputs which can give individual estimations of parameter drift. If two parameters are drifting at the same time, or if there are interdependencies, a fuzzy interpreter can decide which pretrained child neural network is better to transition to, one advocated by diagnostic output A (i.e., stiffness) or by diagnostic output B (i.e. inertia). Since a gradient shift in control from a parent network to a child network has been demonstrated as stable, then a fuzzy linear allocation of control signals from various networks can offer a more accurate control.

3.8 Performance Improvement

As a measure of controller effectiveness, the standard deviation of the output error (measured as the difference between the given signal and the pole placement ideal) is measured, employing Equation 1 for *avg*, the average value of the error and Equation 2 for σ_ϵ, the standard deviation:

$$avg_x = \frac{1}{N} \sum_{i=1}^{N} \epsilon_i \ , \qquad (3)$$

$$\sigma_\epsilon = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (\epsilon_i - avg_i)^2} \ . \qquad (4)$$

By comparing the pole placed signal ρ to various control approaches for the 1 to 2 command of the illustrative example,

σ_ϵ(ρ-uncontrolled plant, for comparison)	=	.2749
σ_ϵ(ρ-parent only, no adaptation)	=	.0821
σ_ϵ(ρ-parent sudden switch to child)	=	.0701
σ_ϵ(ρ-parent drifted switch to child)	=	.0691
σ_ϵ(ρ-child only, assuming prior knowledge)	=	.0654

where, due to the closeness of the parent to the child in this case, the sudden and drifted transition to the child controller weights do not vary significantly.

For these experiments, parameter shift diagnosis was determined for a command given to a steady state value for demonstration purposes. Diagnosis could be achieved as soon as the signal settled to within the neighborhood of the reference (roughly 0.25 sec).

3.9 Frequency Considerations

A motor under varying inertial loads will experience a change in frequency in the uncontrolled plant, and if diagnoses were being done on an uncontrolled plant, a standard or neural Fourier transform could be taken of the response and used as a diagnostic tool for determining the new inertia. In Kazlas,[24] however, individual fault detection with a neural Fourier was not as successful due to spectral similarity. The analog implementation of a neural Fourier transform experienced additional problems with approximation of the absolute value function and when sigmoids were driven into a nonlinear region. Despite these difficulties, frequency domain is still a possible consideration as a voting contribution to diagnosis. A system already under a robust control is difficult to diagnose, however, as the overshoot and settling time of the output signal are brought into closer synchronization. With only a reasonable steady state error in the output to correct for, characteristic magnitude and frequency for the inertias and frictions studied were lost in the signal to noise ratio. Arm twist stiffness, however, benefitted from the explicit frequency domain information.

3.10 Conclusions

Use of an enhanced backpropagation learning algorithm[24] to train a network with two hidden unit layers was adequate for learning the motor relationships for the expected parameter variations. By constraining drifted parameters child controllers to be a gradient transition from the parent, errors that result by mixing competing conventions (i.e. controllers that accomplish a similar result via vastly different combinations of weights) is avoided.[12] If a nonlinear relationship is not learned in a reasonable amount of time, it has been suggested[25] that one could apply a cost function to tune the slope of the sigmoids within the neurons themselves. This may be utilized as the proposed technique is extended to the control of the motors as part of a coupled redundant system.

Instantaneous disturbances affect the short term behavior of a system, which can confuse local estimation schemes. The proposed network searches for a recognizable trend before adjusting the controller, resulting in a pause in highly accurate performance, but avoiding a cyclic oscillation of controller choices. An additional supervisory layer could be added in the future to recognize the effects of some parameter shifts on more than one diagnostic output. By recognizing the changes in distinct features of a signal, a diagnostic network might aid the efforts of other researchers predicting signals that are locally chaotic but globally structured such as music, speech, or power line resonance. Study of the local weight changes of child networks could aid understanding of the way a particular neural network learns the dynamics of the system under study.

References

[1]Fijany, A., and A.K. Bejczy, "A Class of Parallel Algorithms for Computation of the Manipulator Inertia Matrix," **IEEE Transactions on Robotics and Automation**, vol. 5, pp. 600-615, 1989.

[2]Kuperstein, M, "Adaptive Visual Motor Coordination in Multijoint Robots Using Parallel Architecture," **Proceedings of the IEEE International Conference on Robotics and Automation**, p. 1595-1602, 1987.

[3]Massone, L. and E. Bizzi, "Generation of Limb Trajectories with a Sequential Network," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, Vol II, p. 345, 1988.

[4]Yeung, Dit-yan, **Handling Dimensionality and Nonlinearity in Connectionist Learning**. PhD Thesis: University of Southern California, December, 1989.

[5]Miller, D. and E. Davison, "Adaptive Control of a Family of Plants," **Control of Uncertain Systems**, ed. by D. Hinrichsen and B. Martensson. Boston: Birkhauser, 1990. pp. 197-219.

[6]Elsley, R.K. (from Rockwell), "A Learning Architecture for Control Based on Back-propagation Neural Networks," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, 1988. p. II-587.

[7]Josin, G., D. Charney, and D. White, "Robot Control Using Neural Networks," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, 1988. II-625.

[8]Suddarth, S., S. Sutton, and A. Holden, "A Symbolic-Neural Method for Solving Control Problems," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, 1988. p. I-516.

[9]Abu-Mostafa, Y.S., Learning From Hints, speech given at Connectionist Models Summer School, Boulder, Colorado, Tuesday, June 29, 1993.

[10]Kazlas, P., P. Monsen, M LeBlanc, "Neural Network-Based Helicopter Gearbox Health Monitoring System," **Neural Networks for Signal Processing III: Proceedings of the 1993 IEEE Workshop**, ed. by Kamm et al, New York: IEEE,1993.

[11]Narendra, K.S. and A.M. Annaswamy, **Stable Adaptive Systems.** Englewood Cliffs, NJ: Prentice Hall, 1989.

[12]Tascillo, A., V. Skormin, and N. Bourbakis, "Neurofuzzy Grasp Control of a Robotic Hand," **Neural Networks for Signal Processing III: Proceedings of the 1993 IEEE-SP Workshop**, eds. Kamm, Kuhn, Yoon, Chellappa, and Kung, p. 507-516, September 1993.

[13]Tascillo, A. and V. Skormin, "Neural Adaptive Control of Systems with Drifting Parameters," **Proceedings of the 1993 Connectionist Models Summer School**, eds. Mozer, Smolensky, Touretsky, Elman, and Weigend, p. 280-287, 1993.

[14]Tascillo, A.,"Diagnostic Neural Adaptive Control of Drifting Systems," International Journal of Intelligent and Robotic Systems, Kluwer Academic Publishers, to appear in 1995.

[15]Kawato, M., Y. Uno, M. Isobe, and R. Suzuki, "A hierarchical model for voluntary movement and its application to robotics," **IEEE International Conference on Neural Networks**, San Diego, CA, 1987. pp. 573-582.

[16]Wang, H., T.T. Lee, and W.A. Gruver, "A Neuromorphic Controller for a Three-Link Biped Robot, **IEEE Transactions on Systems, Man, and Cybernetics**, Vol. 22, No. 1, January/February 1992.

[17]Yeung, Dit-yan, **Handling Dimensionality and Nonlinearity in Connectionist Learning**. PhD Thesis: University of Southern California, December, 1989.

[18]Yamaguchi, T., K. Goto, M. Yoshida, and T. Mita, "Fuzzy Associative Memory System and its Application to a Helicopter Control," **Fuzzy Engineering Toward Human Friendly Systems**, ed. by T. Terano, M. Sugeno, M. Mukaidono, and K. Shigemasu. Washington, DC: IOS Press, 1992.

[19]Simons, A., **Robotic Trajectory Control Employing Anyanet Neural Architecture**, Master's Thesis: Rensselaer Polytechnic Institute, Troy, New York, 1992.

[20]Goldberg, K., and B. Perlmutter, **Using a neural network to learn the Dynamics of the CMU Direct-Drive Arm II**, Report CMU-CS-88-160, Department of Computer Science, Carnegie Mellon University, 1988.

[21]Jordan, M., and R. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm," in press, **Neural Computation**.

[22]Pomerleau, D. A., "Input Reconstruction Reliability Estimation," **Advances in Neural Information Processing Systems 5**, ed. by C. Giles, S. Hanson, and J. Cowan. San Mateo, Ca.: Morgan Kaufmann, 1993.

[23]Kornhauser, A. L., and E. C. Huber, "Lateral Control of Highway Vehicles Using Image Processing and Neural Networks," **Proceedings of the 2nd Regional Conference on Control Systems NJIT**, New Jersey, August 1993.

[24]Kazlas, P., P. Monsen, and M. LeBlanc, "Neural Network-Based Helicopter Gearbox Health Monitoring System," **Neural Networks for Signal Processing III, Proceedings of the 1993 IEEE-SP Workshop**, Linthicum Heights, Maryland, September 6-9, 1993.

[25]Yamada, T., and T. Yabuta, "Neural Network Controller Using Autotuning Method for Nonlinear Functions," **IEEE Transactions on Neural Networks**, Vol. 3, No. 4, July 1992.

CHAPTER 4:

DYNAMIC REDUNDANT AXIS ROBOT CONTROL

4.1 Traditional Robot Control Approaches

Robot control methods can vary from a simple servomechanism to advanced control schemes such as adaptive control with an identification algorithm. Some popular approaches are joint motion control, resolved motion control, and adaptive control.[1-7] Most of the joint motion and resolved motion control methods[9] servo the arm at the hand or the joint level and emphasize nonlinear compensations of the coupling forces among the various joints. Resolved motion rate control also breaks down near joint space singularities (i.e, when the tool configuration matrix loses rank), so imaginary joints must be added which optimize some criterion chosen by the user.

Model reference adaptive control is easy to implement, but suitable reference models are difficult to choose and it is difficult to establish any stability analysis of the controlled system. Self-tuning adaptive control fits the input-output data of the system with an autoregressive model. Both methods neglect the coupling forces between the joints which may be severe for manipulators with rotary joints. Adaptive control using perturbation theory may be more appropriate for various manipulators because it takes all the interaction forces between the joints into consideration, but it can be computationally intensive and its assumptions may not be applicable for all situations. Some traditional approaches applicable to this work's emphasis on adaptation and redundancy are discussed below, followed by researchers who focused on a neural network solution to the robotic control problem.

Wu, C., and C. Jou[8], while designing a controlled spatial trajectory for robot manipulators, present a two-phase robot trajectory planning method that decomposes the trajectory planning problem into path geometry planning and robot movement speed planning. The desired path is maintained while the speed is varied. The path geometry can be used to verify against geometric and kinematic constraints of the robot manipulator, while the speed function of robot motion can be used to verify against dynamic constraints.

Walker, et.al.[9], while investigating robustness issues for kinematically redundant manipulator control, predict performance of rigid manipulator controllers without complete parameter knowledge. The method introduces redundancy as part of the manipulator states by inserting a parameter matrix P with upper and lower bounds, and its estimate P_{hat}, for the equation of motion:

$$A(P,\theta)\ \ddot{\theta} + V(P,\theta,\dot{\theta}) + G(P,\theta) = \tau^{C}, where \qquad (1)$$

θ = set of rigid and flex degrees of freedom (n x 1),
A = the symmetric, positive definite mass matrix (n x n),
V = centripetal and coriolis torques (n x 1),
G = gravitational torque (n x 1), and
τ^{C} = control torque (n x 1).

The computed torque controller for this system was then chosen as:

$$\tau^C = A(\hat{P},\theta)\ (\ddot{\theta}_d + K_d\, \dot{e} + K_p\, e) + V(\hat{P},\theta,\dot{\theta}) + G(\hat{P},\theta)\text{, where} \quad (2)$$

θ_d = desired joint accelerations,
e = θ_d - θ = joint position errors,
derivatives of the terms in the above equation for joint velocity errors, and
K_d,K_p = diagonal feedback gain matrices.

This type of controller is currently the most widely used type of control in robotics. It is popular due to its simplicity, essentially linearizing the dynamics and applying PD control at each joint. However, the approach requires perfect cancellation of the terms V and G, and calculation of the matrix A. Poor tracking and stability can follow. An adaptive perturbation control strategy was found by the researchers to be suitable for controlling the manipulator in both the joint coordinates and cartesian coordinates. An adaptive perturbation control system is characterized by a feedforward component and a feedback component which can be computed separately and simultaneously in parallel. The computations of the adaptive control for a six-link robot arm may be implemented in low-cost microprocessors for controlling in the joint variable space, while the resolved motion adaptive control cannot be implemented in present-day low-cost microprocessors because they still do not have the required speed to compute the controller parameters for the "standard" 60-Hz sampling frequency.

Given the limitations of traditional approaches, a method is desired that does not specifically require a matrix representation, and therefore does not also go unstable near joint singularities. The author's contribution[10,11] is the design of a joint allocation network that suggests a weighted contribution of a particular joint to a task given its perceived state of health or stress. This network is trained with the outputs of optimization equations whose goals often conflict. The approach is demonstrated in simulation for the complete redundant robot.

4.2 Neural Network Approaches

Not much work has been done with neural networks for combined trajectory following and dynamics for five, six, and redundantly jointed robots. Most researchers stop at two or three joints[i.e. 12-17] and the vast majority learn inverse kinematics,[i.e. 12-14,18] not dynamics. Of researchers who have extended to a larger number of joints, the controllers generally consist of a neural network map of a robot's workspace, which converts to motor movements or joint angles necessary to track to the object location indicated, essentially memorizing the robot's inverse kinematics. The most successful dynamics work[15] has involved linear combinations of conventionally estimated parts of the robot dynamics equation.[15-17] The linear combinations could be varied on line to adapt to changes due to load, and were accurate for their specific robot, but Kawato et. al.'s configuration could not be applied to any another robot or modification to that robot. This method may not be acceptable over time or with injury to components. Some insights can be gained from looking at these applications in more detail.

The most successful neural network dynamics work by Kawato et.al.[15] has involved one layer linear combinations of conventionally estimated parts of the robot dynamics equation

q(t) is an nx1 vector of joint linear/angular positions,
I(q,t), an nxn matrix of terms related to inertial forces

$$T(t) = I(q,t)\,\ddot{q}(t) + H(q,\dot{q},t)\,\dot{q} + B\,\dot{q}(t) + G(q,t), \text{where} \quad (3)$$

H(q,q,t), an nx1 vector of centripetal & Coriolis forces,
B, an nxn diagonal matrix of viscous friction terms,
G(q,t), an n x 1 vector of gravitational force terms,
T(t) the n x 1 vector of driving forces or torques, and
n, the number of degrees of freedom of the manipulator.

The control performs well, but is specific to the manipulator it was designed for. In one paper, a recursive Jordan network (some initial nodes employ their previous value as another input) was used to control a two link planar manipulator[12]. Their model differed from Kawato's dynamic controller in that he studied voluntary movement and proposed a hierarchical, structured model for generating motor commands (torques) from a desired trajectory expressed in body centered coordinates. Moreover, he studied the coordinate transformation problem and proposed an iterative control learning algorithm. This research, on the other hand, dealt with a sensory-motor transformation based on a non-hierarchical layered architecture translating sensory stimuli directly into time-varying patterns of muscular activation corresponding to minimum jerk trajectories. A coordinate transformation problem was avoided by hypothesizing that both target and movement were already expressed in the same body-centered reference frame.

A kinematics controller was trained in a second paper for another planar two joint arm[13], whose equations were:

$$\theta_1 = \tan^{-1}(y/x) - \tan(k_2-k_1) \quad \text{and} \quad \theta_2 = \tan^{-1}(S_2/C_2),$$

where

$$k_1 = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2),$$
$$k_2 = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2),$$
$$C_2 = \cos(\theta_2) = (x^2 + y^2 - L_1^2 - L_2^2)/(2\ L_1\ L_2), \text{ and}$$
$$S_2 = \sin(\theta_2) = \pm\ \mathrm{sqrt}(1 - C_2)$$

The four inputs to the neural network were desired x and y position in addition to the θ_1 and θ_2 suggested by the regular controller. There were 24 units in one hidden layer. Outputs were $\nabla\theta_1$ and $\nabla\theta_2$ which were added to the regular controller suggested angles θ_1 and θ_2. After training on a set of targets, the base joint was stiffened for the first error, and the second joint was bent for the second error. For training on test points in and out of the original training region, the neural controller compensated for 60% of the error in both error cases after experiencing just one target point. After three points, accuracy improved six times, and after 8 points learning leveled off with an 11 times improvement over the original controller accuracy. These results indicate that a neural network is successful in learning sinusoidal relationships for a planar two joint arm, and can demonstrate online adaptation when new areas of the workspace are introduced. It is unclear if this direct learning will be stable in situations of parameter drift or for larger numbers of joints.

In a third paper inverse kinematics was learned for a planar two degree of freedom robot[14]. The internal state of the system and error in position formed the inputs to a neural network which was trained to be an inverse Jacobian accurate for long trajectories. The classical inverse Jacobian is only accurate for moves in the linear neighborhood of the current state. The actual control input to the system was the network output multiplied by the position error of the system. The neural network itself had two voltage inputs, each of which lead to their own four overlapping gaussian functions. Groups of two gaussian neurons were multiplied together at one of the eight cartesian neurons. All cartesian neurons fully connected with each of the eight sigmoidal learning neurons. These last two layers were modified via backpropagation during learning. Each of the two voltage output neurons in the final layer were connected to their own four learning neurons. The output neuron activation function was the first moment of the states of the learning layer neurons.

The learning rate during online adaptation was adjusted proportionally to the position error. Also, if the network suggested no correction for a currently existing position error, a nonzero learning input was given to the neural net causing motion to continue. Training was accomplished online, where the goal, being a two dimensional point in space, was presented a maximum of 15 times as the network attempted to reduce the error through adjustment of its inverse Jacobian weights. After roughly 25 goal presentations the robot required a maximum of two steps to attain the goal. After 500-1000 additional goals were presented, an average error of approximately 6% resulted. Another test consisted of removing 10% of the trained weights, and then retraining the remaining weights. The error doubled, but then returned to nearly the previous level. An additional 10% of connections were removed twice more, resulting in errors of 40 and

50%. The final error returned to nearly 10% each time. This paper demonstrates a successful mixing of neuron types, and shows the robustness of the controller despite damage to its weights.

Moving to a paper whose controller accounted for dynamics, a neural network motor controller was added to a K gain multiplier of the system error[15]. The network consisted of linear combinations of the components of the inverse dynamics equation. This work is an earlier version of an application explained in the following article summary.

The control signal for a three joint manipulator[16] was synthesized by adding the control command of a PD controller to the suggested adjustment offered by a linear combination of subnets, each representing a known component of the given robot's dynamics. First, the adjustable final layer linear weights were initialized at zero, allowing only feedback control. Learning adjustment of these weights then required 30 minutes of a 6 second pattern repeated 300 times with sample and update rate placed at 100 Hz (sample step 0.01 seconds). The trained system remained stable under load, and when learning commenced, the steady state error was eliminated as before. Both of these papers used linear combinations of components of a dynamics equation to demonstrate their concepts.

In a dissertation that touched on online adaptive control of a DC motor, the author himself[17] concedes that, "our study focuses more on the capabilities of neural networks in learning inverse models than on the design of feedback control systems with feedforward control. The latter concern is an issue on its own." He does, however, have some valuable comments to make on other researchers, summarized below. Kuperstein[18] attempted to achieve visual-motor coordination by relating the sensory maps directly with the motor commands in order to reach some specified target locations. Although these methods have been demonstrated to work for robot arms with two or three degrees of freedom (DOF), they may not perform equally well for 6-DOF arms. In fact, Yeung's attempts to apply these methods directly to more realistic robot arms with a larger number of DOFs were "discouraging." Kawato[8] overcomes this problem with linear combinations of simpler functions, but the dynamic equation must be known in advance. Kawato's feedforward/feedback control, with only one layer of weights updated online via the guaranteed convergence Widrow-Hoff rule, controlled well. Yeung's learning procedure may lead to instability if used in an online fashion due to more than one layer of adjustable weights in the modifiable context network.

In a paper concerning coordination of two "planar" manipulators, a hierarchy of neural networks was used to control walking in a biped robot[19]. Nonlinear feedback decoupling was utilized for training. Competing hypotheses of modular subfunction nets were combined for robustness. Robust to parameter variations and disturbances, this controller improved its performance through learning. The training control model was based upon nonlinear feedback decoupling and optimal tracking. A critic network added an additional input to the *supnets*, or task networks, which were themselves composed of time sequenced *subnets*. Example supnet tasks specialized in leg support control, regulation of body orientation, and leg swing control. The 5 subnets split the walking task up into phases that were more easily learned by a three input, 20 sigmoidal unit first layer, 3 sigmoidal unit second layer, and one scaled linear output unit for actuator control. 20 patterns were used to train the network, so the first hidden layer consisted of 20 units to memorize the patterns. The three inputs were position, angular velocity, and desired position. 10,000-100,000 epochs were required for learning with adjustable learning rate and momentum. Performance after learning (demonstrated after 450 20-millisecond samples, or 9 seconds) equaled or surpassed both quantized and continuous PID controllers for various tracking problems. The author claims guaranteed absolute stability, but the effects of friction, dynamic aspects of the foot, leg to leg transfer and non relevant planar motion were ignored for simplicity.

4.3 Redundant Allocation of Joints

For this effort the individual joint diagnostic motor controller outputs discussed in Chapter 3 are interpreted by weight allocation neural networks (Figure 1), whose task is to propose a proportion of a motion a joint feels capable of performing. Following the lead of the literature by assuming rigid joints, the proposed redundant manipulator allocation method will allow each individual joint controller to handle the task of adapting to the nonlinear effects it sees at a local level. The effort it requires to overcome these nonlinearities will be reflected in the differential values of the diagnostic outputs.

The weight allocation neural network suggested proportions are interpreted via fuzzy logic to determine the relative state of health or excessive stress for joints considered equal contributors to a component of desired motion. If the results are inconclusive, an expert system determines from experience what initial weights to start with given data about the similarity of a new situation to others it has seen before.

If a hand joint redundant to a main robot joint does exist, then its weight is added to the chosen weight and the motor contributions are

$$\frac{w_{base}}{w_{base} + w_{hand}} \qquad (4)$$

where w_{base} is the main robot minimum recommended weight and w_{hand} is the relevant hand weight. The inverse kinematics are then solved for that proportion of the necessary motion and the hand is asked to complete the movement. For example, assume that joints 1 and 7 of the redundant robot base and hand combination are determined redundant for yaw, and A is the total current expended for the movement. If the critical weight determined by the neurofuzzy weight allocation algorithm is that for the hand joint 7, then the final proportion of effort will be allocated

$$w_7 * A$$

for joint 7, and

$$(1-w_7) * A$$

for the redundant joint 1.

Since the configurations all satisfy acceptable kinematic constraints, all are stable. One configuration is simply more optimal from a longer time horizon standpoint (i.e. anticipates greater accuracy in the future).

4.4 Choice of Cost Functions

The goal of redundant weight allocation is to best implement the planned trajectory of higher level planning. First, the next via point of a trajectory is chosen by a method such as the Generalized Voronoi Diagram (GVD)[5], which is commonly used to map out the shortest path to a target equidistant from obstacles. Then the challenge is to find the sequence of joint reference commands that optimize a cost function as a robot manipulates an object within the workspace. This cost function would ideally retain the minimum overshoot and minimum settling time goals of the joints despite disturbances, not only due to changes in loading, friction, and beam stiffness, but also due to the effects of gross positioning of the manipulator.

These costs functions should also be able to accommodate useful rules of thumb. For instance, the three large joints nearest to the base (proximal) are better at gross motion and attainment of manipulator cartesian position. Although this rule at first would suggest that the algorithm will allocate most of the gross motions at any given moment to the largest joints, they also cannot move as quickly, and will be stressed when a fast movement is required. Complicating these two constraints, priority should be given to the most proximal angle for successful avoidance of moving obstacles whose point of closest approach has not yet been determined. If the base joints are maintained close to the center of their operating ranges, they are better prepared for large distance evasive action.

Further constraining the choice of joint allocation are those passed down from the object tip and slip algorithms of Chapter 5. For instance, the current orientation of an object must be closely maintained in order to avoid spills and other instabilities due to the change of center of mass. This constraint can often simplifies number of joints to essentially six during transfer of the object across the workspace once stably grasped. The cost of moving the hand joints should be weighted higher than the cost of maintaining the current orientation. During the time that tip and slip algorithms are active, their motion takes priority at the hand joint level, and priority must be given to motions that cause minimal object relative motion.

By repeatedly reassessing the costs of particular allocations of the redundant joints, one can optimize over a smaller range of movement, in a fashion similar to that of Kim and Shin.[20] They developed a suboptimal method for controlling the first three links of a PUMA manipulator (with DC servomotors) that provides improved performance in both operating speed and use of energy. The nonlinearity and the joint couplings in the manipulator dynamics are handled by averaging the dynamics at each sampling interval. With the averaged dynamics, a feedback controller is derived which has a simple structure allowing for online implementation with inexpensive computers, and offers a near minimum time-fuel (NMTF) solution, thus enabling the manipulator to perform nearly up to its maximum capability and efficiency. The NTBF controller derived in the paper, however, is concerned only with free space motion, or noncompliant motion. For the compliant case, the dynamic equation becomes far more complex; for example, the static friction terms cannot be ignored, but cannot be modeled accurately, either. With an accurate dynamic model of the manipulator for the compliant case, and with a suitable performance criterion[21] an optimal controller could be derived.

Following are the cost functions[22] chosen to be summed as targets for the neural joint allocation networks to learn. The first function is the standard minimum time/minimum distance of travel,

$$J = N = \sum_{k=0}^{N-1} 1 \, , \quad x_n = x \tag{5}$$

This basic equation has to be limited to avoid oscillations and range/rate limits. The second function, minimum fuel,

$$J = N = \sum_{k=0}^{N-1} |u_k| \,, \quad x_n = x \tag{6}$$

seeks to maintain the status quo over sudden changes in joint position. An ailing motor is given even higher weighting for minimal usage. The third function, minimal energy,

$$J = \frac{1}{2}\, x_N^T\, S\, x_N + \frac{1}{2} \sum_{k=0}^{N-1} (x_k^T\, Q\, x_k + u_k^T\, R\, u_k) \tag{7}$$

favors the center of the earth, assuming 90 degrees is down, as well as the absolute value of the current state minus (90/57.3) to maintain midrange for future movement flexibility. The fourth function, minimum jerk[12],

$$J = \left| \dddot{q} \right| \,, \text{ where } q = \text{joint position,} \tag{8}$$

minimizes unnecessary harsh movements during acceleration and deceleration. The fifth function, minimum coupling, minimizes the effects of one joint of the positioning of others, specifically those joints that are redundant. After allocating favoritism due to ailing or stressed joints, priority allocation is given to the most proximal redundant joint. Functionally, this fifth cost can be expressed as

$$J = [R_{ij} \in \frac{1}{2} \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k)] , \tag{9}$$

or in other words, minimization of the off diagonal elements of the R matrix or its equivalent. Coupling also must account for self motions, when two or more axes of a robot form a straight line, that is, become collinear. Here the effects of rotation can be canceled by a counteracting rotation about the other axis. The above functions are supplemented with the aforementioned rules of thumb and constraints.

4.5 Dynamic Simulation of the Redundant Robot

Simulation of the redundant PUMA base robot plus hand was accomplished in Simulink with a sampling frequency of 1 MHz to closely approximate the analog system, necessary to avoid simulation instability. Figure 2 shows the overall simulation and Figure 3 is a closeup of one joint. Simulation of the resulting motion with the hand is demonstrated in the language C in the remaining figures.

When a new desired position was determined for a given joint, a ramp command was given from the previous state to the new state in order to avoid exciting resonant frequencies similar to those experienced by Tarn, et.al.[23], who excited joints 2 and 3 of the Puma at 1350 Hz and the other joints at 1600 Hz. They found that an acceptable

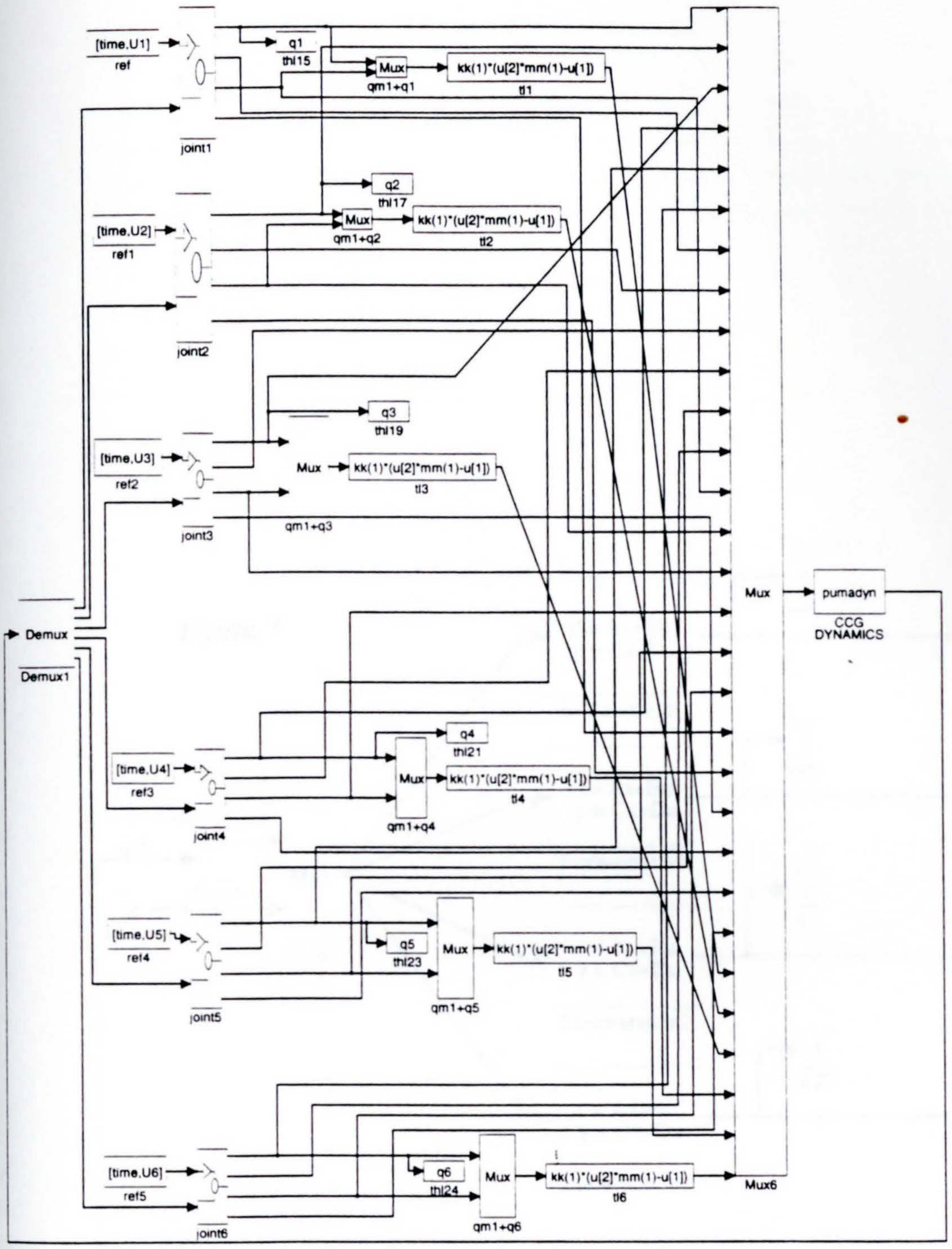
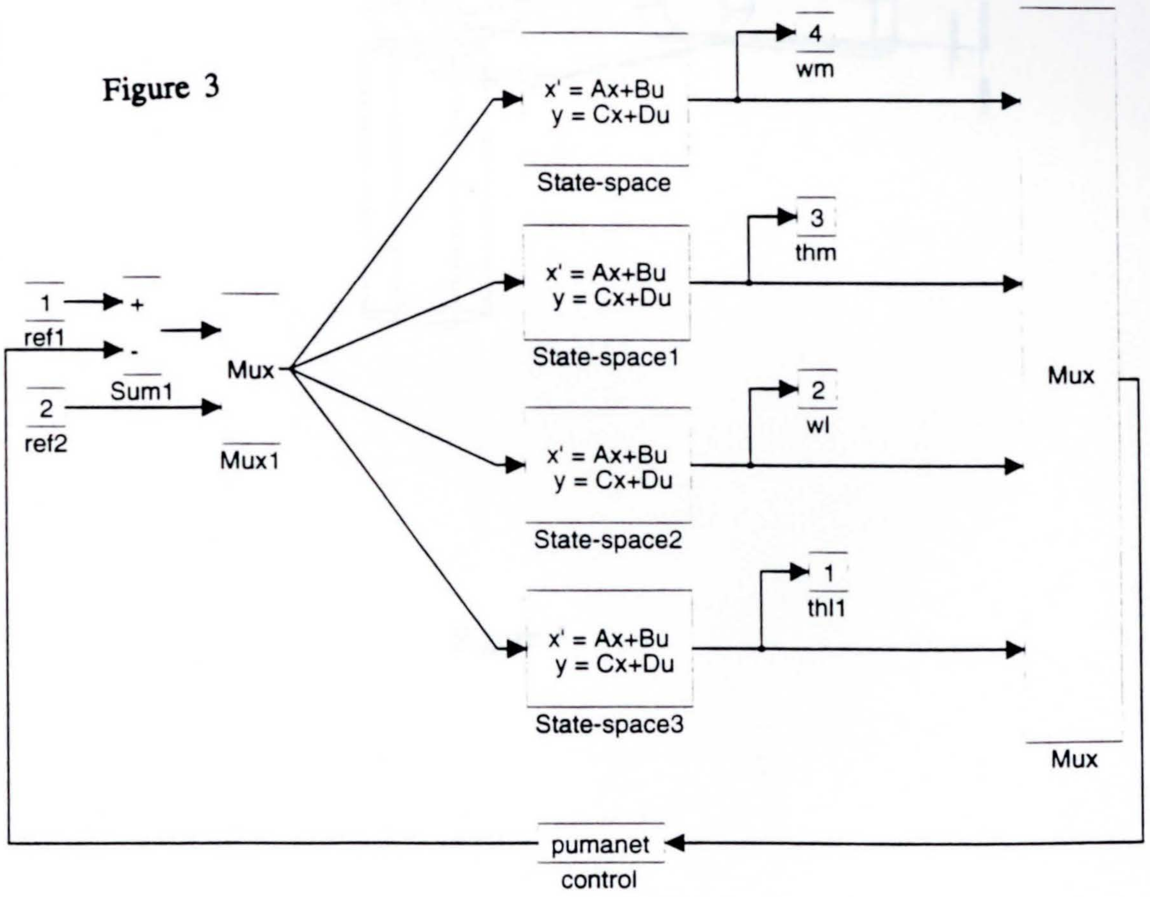

[time,U1]
ref
q1
thl15
Mux
qm1+q1
kk(1)*(u[2]*mm(1)-u[1])
tl1
joint1
q2
thl17
[time,U2]
ref1
qm1+q2
tl2
joint2
q3
thl19
[time,U3]
ref2
tl3
joint3
qm1+q3
Demux
Demux1
q4
thl21
[time,U4]
ref3
qm1+q4
tl4
joint4
[time,U5]
ref4
q5
thl23
qm1+q5
tl5
joint5
[time,U6]
ref5
q6
thl24
qm1+q6
tl6
joint6
Mux6
pumadyn
CCG DYNAMICS

Figure 2

Figure 3



1
ref1
+
-
Sum1
2
ref2
Mux
Mux1
x' = Ax+Bu
y = Cx+Du
State-space
State-space1
State-space2
State-space3
4
wm
3
thm
2
wl
1
thl1
Mux
Mux
pumanet
control

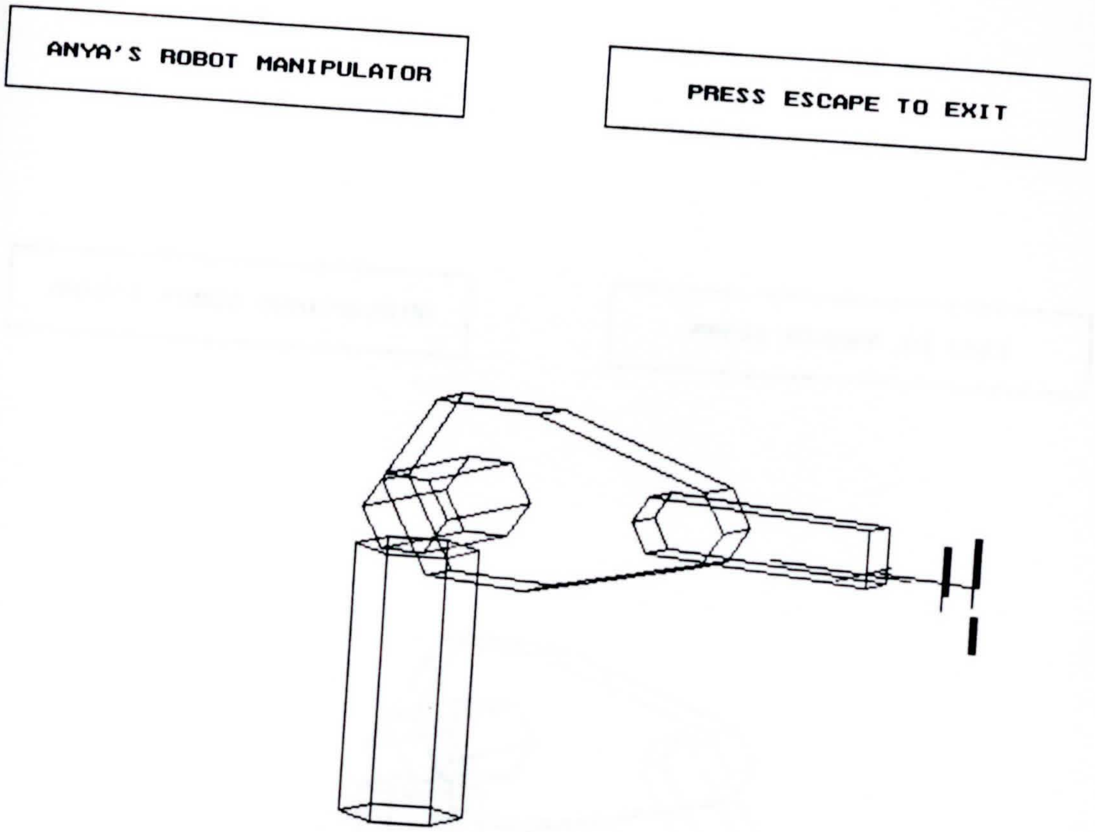


Figure 4

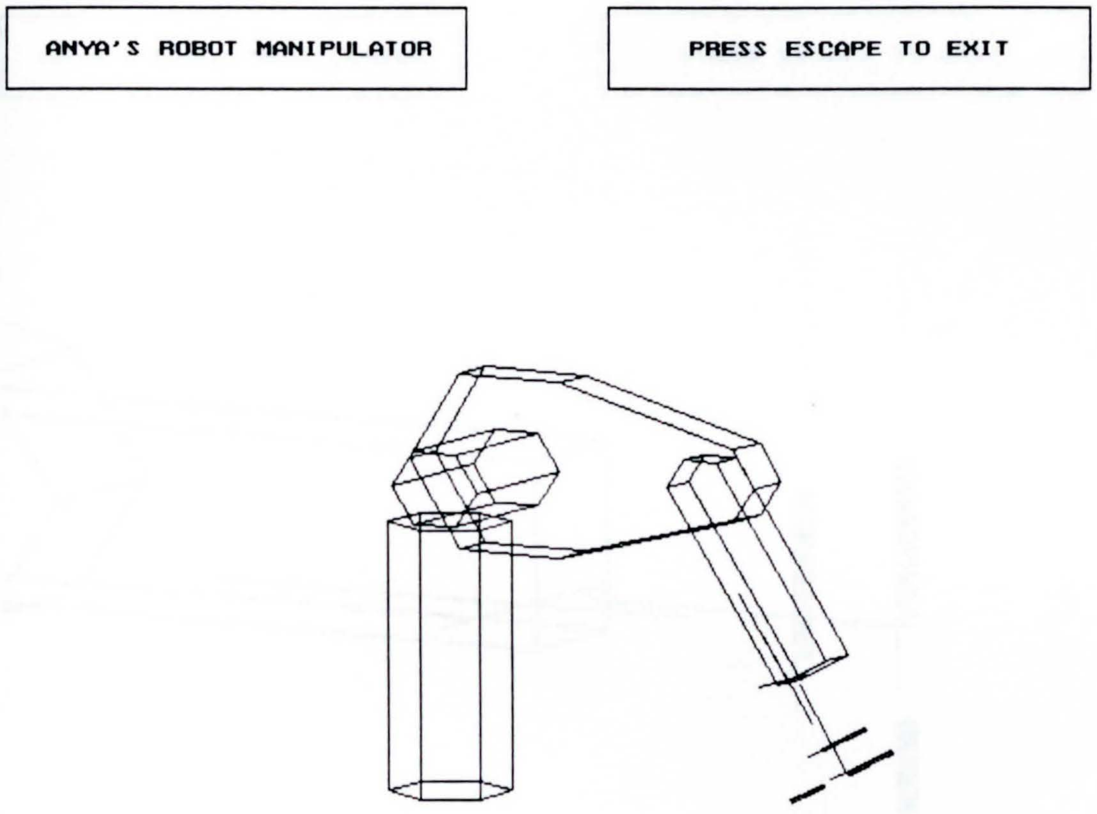


Figure 5

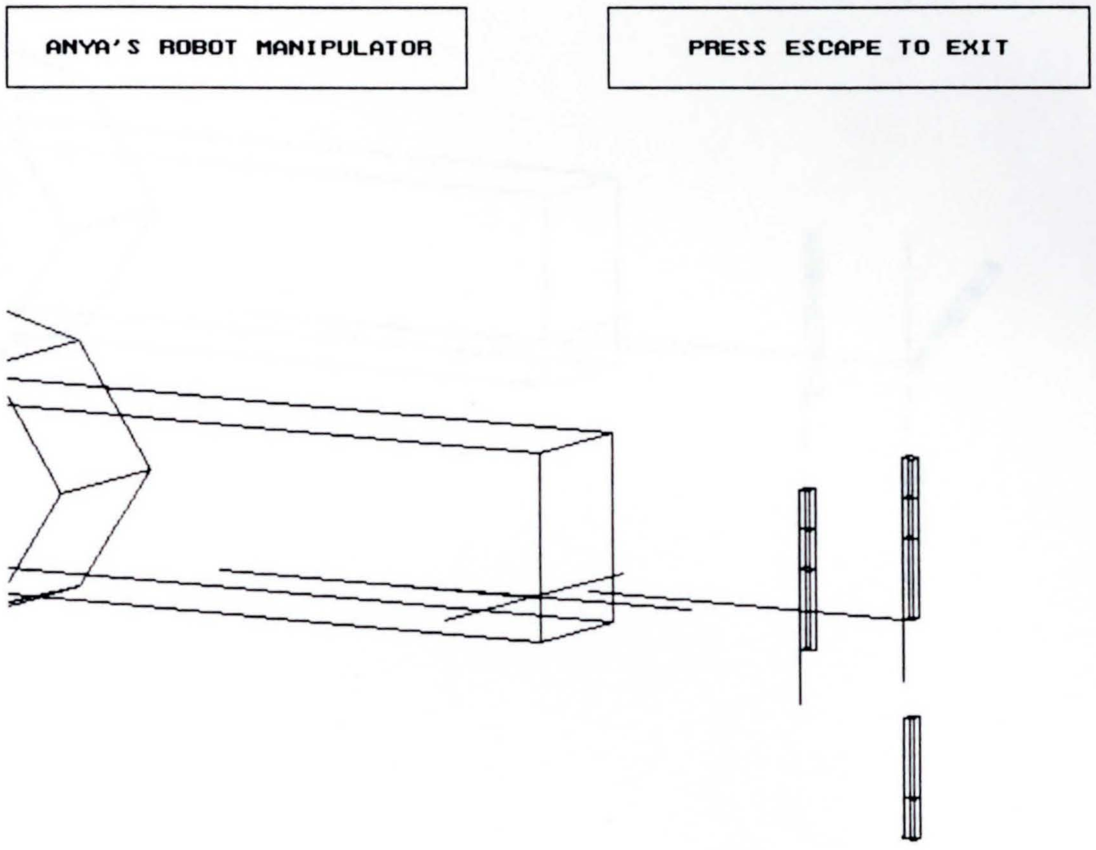


Figure 6

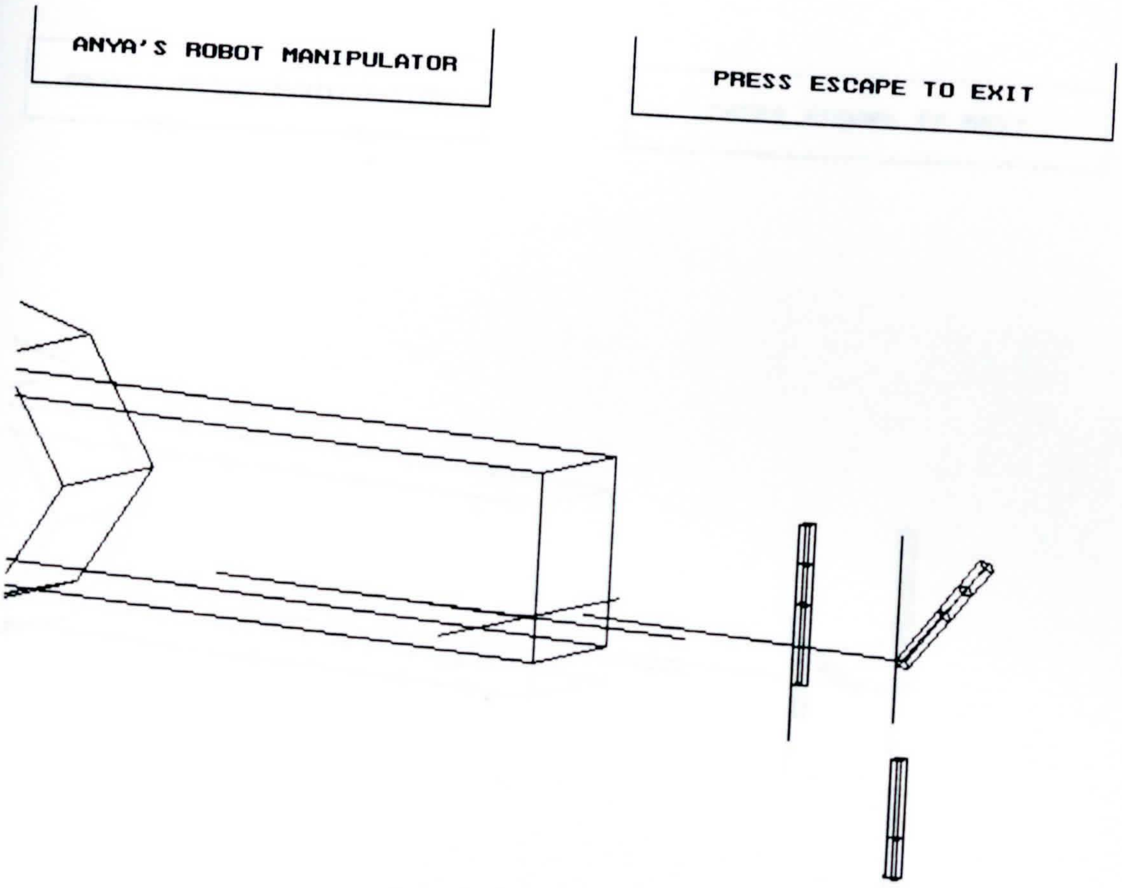


Figure 7

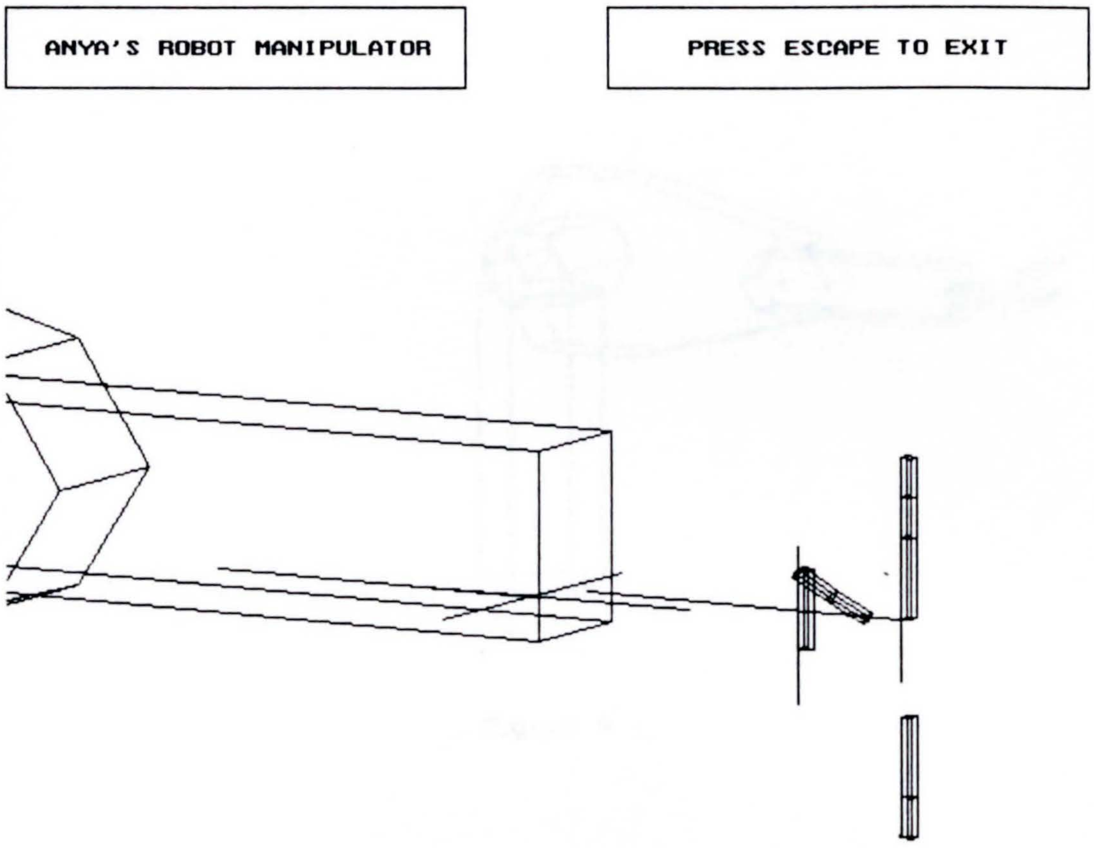


Figure 8

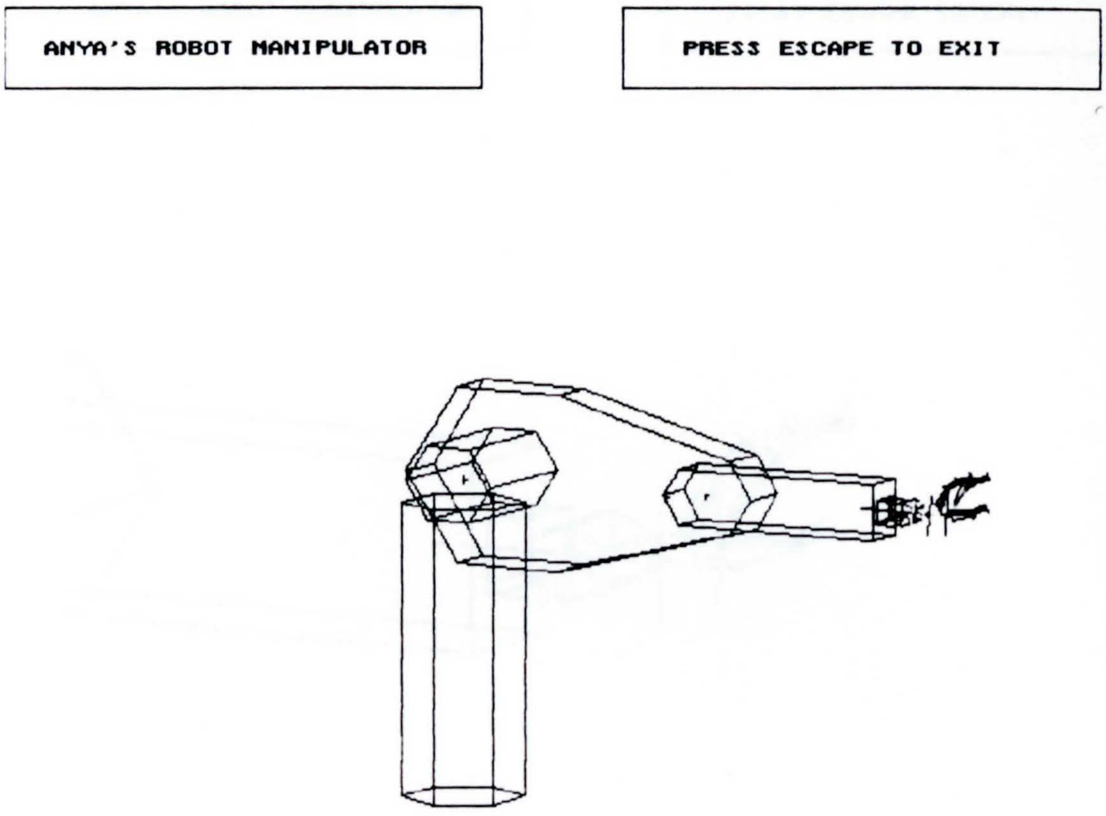


Figure 9

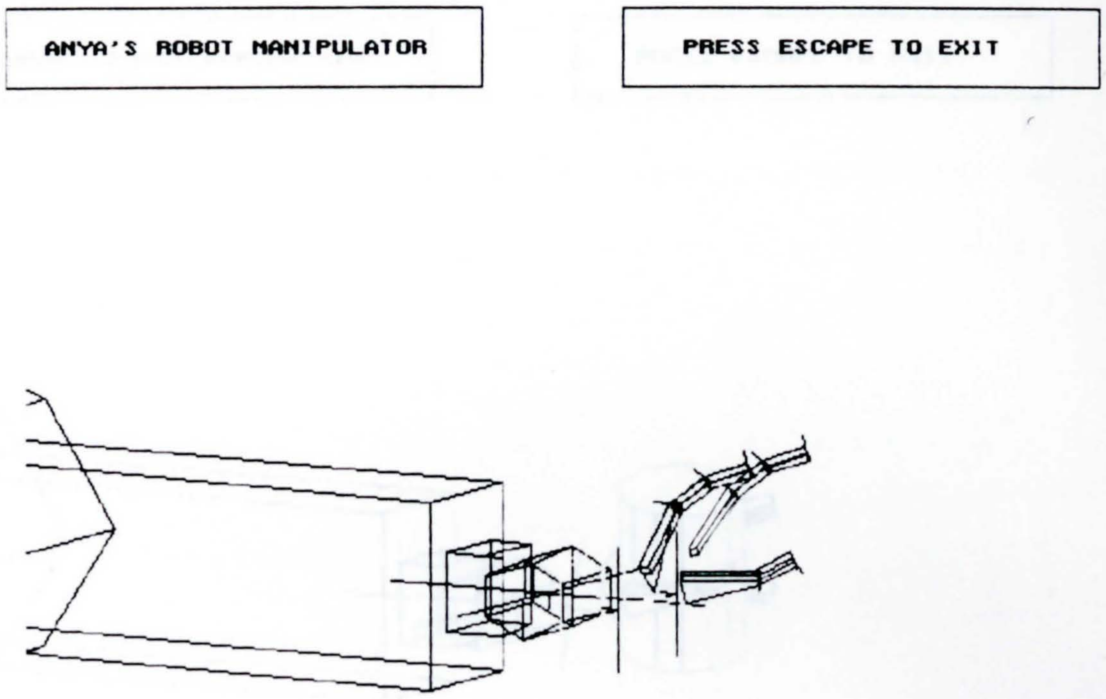


Figure 10

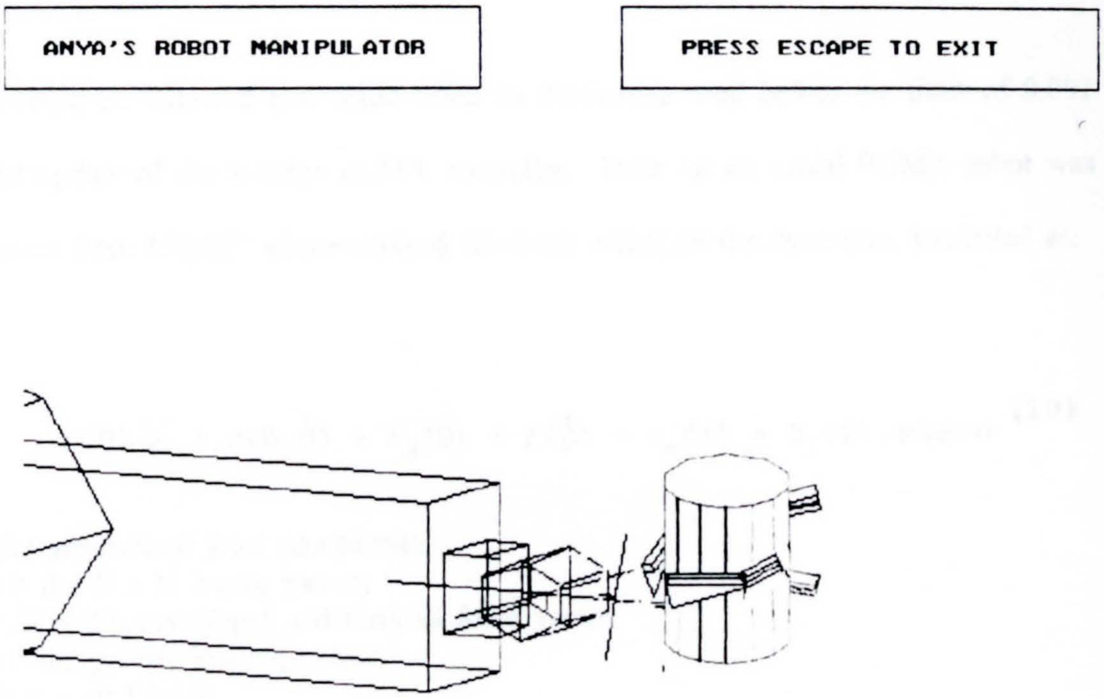


Figure 11

ramp could be followed that could reach its destination well before the limit of 0.001 second update of the average PUMA controller. Data for an actual PUMA robot was borrowed from Lloyd,[24] who examined friction's effect on the dynamics, modelled as:

$$D(\theta)\ddot{\theta} + c(\theta,\dot{\theta}) + t_g(\theta) + f(\dot{\theta}) + t_a(t) = t_j(t), where \tag{10}$$

θ = the generalized joint coordinates,
$D(\theta)$ = the N x N inertia matrix,
$c(\theta^{dot},\theta)$ = the centrifugal and Coriolis force vector,
$t_g(\theta)$ = the gravity loading vector,
$f(\theta^{dot})$ = joint friction,
$t_a(t)$ = externally applied joint forces, and
$t_j(t)$ = joint torques/forces exerted by the motors, where
$t_j(t) = t_{g(t)}(\theta) + f(\theta^{dot}) + t_a(t)$.

The gravity loading model became:

$$\tau_{g(t)} = \sum_{k=1}^{N} \left(-m_k\, g^T \frac{\delta T_k}{\delta \theta_i} k_{r(k)} \right), where \tag{11}$$

$t_{g(t)}$ = the torque joint i exerts to overcome gravity,
N = the total number of links for the robot,
m_k = the mass of link k,
g = the acceleration of gravity vector (base coord.'s),
T_k = the 4 x 4 transformation matrix (base frame to link k),
θ_i = the joint variable for joint i, and
$k_{r(k)}$= the center of mass for link k, (link coord. frame).

After inserting real measurements of torque sensitivity, friction, and gravity loading, this model determined the net torque acting on a joint as well as the desired net torque for a joint to exert. These measurements were useful for assigning realistic values

[5]Schilling, R.J., **Fundamentals of Robotics: Analysis and Control**. Englewood Cliffs, New Jersey: Prentice Hall, 1990.

[6]Seraji, Lee, and Delpech: Direct Adaptive Dynamic Control., "Experimental Study on Direct Adaptive Control of a PUMA 560 Industrial Robot," **Journal of Robotic Systems**, John Wiley and Sons, 1990. Parameter Adaptation of PD controller. Stiction accounted for.

[7]Lin, L.-C. and K. Yuan, "Control of Flexible Joint Robots via External Linearization Approach," **Journal of Robotic Systems**, 7(1), 1-22, 1990.

[8]Wu, C., and C. Jou, "Design of a Controlled Spatial Trajectory for Robot Manipulators," **Proceedings of the IEEE Conference on Decision and Control**, 27th Conference, Vol. WA6-10:15, Austin, TX, Dec. 1988, p. 161-165.

[9]Walker, I.D., L.A. Nguyen, and R.J.P. De Figueiredo, "Robustness Issues in Kinematically Redundant Manipulator Control," **Proceedings of the 30th Conference on Decision and Control**, Brighton, England, December 1991.

[10]Tascillo, A., V. Skormin, and N. Bourbakis, "Neurofuzzy Grasp Control of a Robotic Hand," **Neural Networks for Signal Processing III: Proceedings of the 1993 IEEE-SP Workshop**, eds. Kamm, Kuhn, Yoon, Chellappa, and Kung, p. 507-516, September 1993, with V. Skormin and N. Bourbakis.

[11]Tascillo, A., "Neural Redundant Robotic Trajectory Optimization with Diagnostic Motor Control," ICNN '94: **Proceedings of the IEEE International Congress on Neural Networks**, June 28-July 2, 1994.

[12]Massone, L. and E. Bizzi, "Generation of Limb Trajectories with a Sequential Network," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, Vol II, p. 345, 1988.

[13]Josin, G., D. Charney, and D. White, "Robot Control Using Neural Networks," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, 1988. II-625.

[14]Elsley, R.K. (from Rockwell), "A Learning Architecture for Control Based on Back-propagation Neural Networks," **IEEE International Conference on Neural Networks**, Sheraton Harbor Island, San Diego, CA. July 24-27, 1988. p. II-587.

[15]Kawato, M., Y. Uno, M. Isobe, and R. Suzuki, "A hierarchical model for voluntary movement and its application to robotics," **IEEE International Conference on Neural Networks**, San Diego, CA, 1987. pp. 573-582.

[16]Miyamoto, H., M. Kawato, T. Setoyama, and R. Suzuki, "Feedback Error Learning Neural Network for Trajectory Control of a Robotic Manipulator," **Neural Networks**, Vol. 1, pp. 251-265, 1988.

[17]Yeung, Dit-yan, **Handling Dimensionality and Nonlinearity in Connectionist Learning**. PhD Thesis: University of Southern California, December, 1989.

[18]Kuperstein, M, "Adaptive Visual Motor Coordination in Multijoint Robots Using Parallel Architecture," **Proceedings of the IEEE International Conference on Robotics and Automation**, p. 1595-1602, 1987.

[19]Wang, H., T.T. Lee, and W.A. Gruver, "A Neuromorphic Controller for a Three-Link Biped Robot, **IEEE Transactions on Systems, Man, and Cybernetics**, Vol. 22, No. 1, January/February 1992.

[20]Kim, B.K., and K.G. Shin, "Suboptimal Control of Industrial Manipulators with a Weighted Minimum Time Fuel Criterion," **IEEE Transactions on Automatic Control**, Vol. AC-30, No. 1, January 1985.

[21]Shin, K.G. and N.D. McKay, "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints," **IEEE Transactions on Automatic Control**, Vol. AC-30, No. 6, June 1985.

[22]Lloyd, John, **Implementation of a Robot Control Development Environment**, ME Thesis, McGill University, December 1985.

[23]Lewis, Frank L., **Optimal Control**. New York: John Wiley and Sons, 1986.

[24]Tarn, T.J, A.K. Bejczy, G.T. Marth, and A.K. Ramadorai, **Kinematic Characterization of the PUMA 560 Manipulator**, Robotics Laboratory Report SSM-RL-92-15, Department of Systems Science and Mathematics, Washington University, Saint Louis, Missouri, 1991.

CHAPTER 5: GRASP PLANNING WITH TIP AND SLIP CONTROL

5.1 Grasping with a Redundant Robot and Hand

Fuzzy logic and neural networks are employed to control a robotic hand with two three-jointed fingers and a two-jointed thumb. The hand algorithm is expected to estimate a grasp for an unknown object, adjust its grasp until one that is most stable is found, and then lift the object, adjusting for tip and slip. Additionally, since vision cannot be depended upon in a cluttered environment such as a refrigerator, the algorithms must be able to operate with little or no support from camera data. Instead, pressure and force feedback are used. An attempt is made here to determine an optimal approach angle and hand orientation for a range of similar objects, with little regard for features that humans might insist on considering (i.e., that a mug has a handle and no other option exists for lifting). For the hand an architecture and algorithm based upon backpropagation was chosen, due to its relative training stability and ability to provide continuous output values.

Before any grasping is to be attempted by a robotic hand, the higher level in the hierarchy devoted to planning will approach an object from a distance away, allowing the localized grasp algorithms to be optimized for more intricate motions. Once in the proximity of the object, however, two challenges arise, "How should the object be approached?", and once grasped, "How can stable and reliable control be maintained?". Hu, et. al.[1] agree that local reference frames aided in the navigation of an autonomous robot via a Kalman filter approach. A combination of both global and local perspectives can provide the control hierarchy with the information it needs to make a decision optimal from both short and long term points of view.

An initial grasp can be estimated simply as a direct computation based upon sensory feedback, or it can draw from past experience based upon object features. One example of the first case can be found in,[2] which employs a frictionless point contact model (best applicable to objects large compared to the fingers) of an object to evaluate quality as a function of the disturbance to the object caused by the fingers. The algorithm assumes small grasp adjustments and is most accurate for objects approaching the size of the hand itself. Prior to this approach one could apply that of Nguyen and Stephanou,[3] who define a grasp as a combination of possible subconfigurations and use a knowledge base to search for an optimal grasp from the initial approximation. If the experience gained from the first approach can be used to add data to the knowledge base, the controller could continue grasp learning unsupervised.

One way to accumulate knowledge could be by matching object features in the database, and adding the experience gained. One example is Nasrabadi and Li,[4] where object profiles, based upon polygonal approximations of two dimensional visual data, are compared with object polygons in a database via a Hopfield network. Classification is successful with objects which are rotated and whose estimated polygon vertices outnumber those of the object in the database. Physical contact could be utilized in a similar way. If the data is hard to match from the sensory data available, offer an enhancement.[5] Because a neural feature extraction network often has problems identifying highly noisy detailed patterns, a pattern is sectioned up into blocks, sub-blocks of which are clustered by dissimilarity and then entered into the neural network. The final layer of the network consists of noise filtering fuzzy membership functions, which compare their results to patterns in a database.

Once the object has been grasped in a perceived optimal manner, it must be lifted successfully, and factors such as tip and slip must be compensated for. Adaptive control is necessary both for grasp stability and for negotiating through cluttered environments. Classical control techniques may not be adequate to compensate for extremely nonlinear behavior. One example of classical feedback control[6] employs finger tip pressure and position feedback control to move a peg from one hole to another. The peg must be dragged along the worksurface between the holes to obtain necessary feedback information. Neural networks and fuzzy logic show promise in tandem with or when compared to classical approaches. In Scharf and Mandic,[7] an adaptive fuzzy robotic trajectory tracker compares favorably to continuous and quantized input PID controllers. In Wang. Lee, and Gruver,[8] a neural network design with nonlinear feedback decoupling is used to control a biped robot that is robust to disturbances. Sub-functions in the walking task are controlled with small dedicated networks.

Given a user's expectation that a grasping robot will learn from experiences with similar objects, the author's contribution[10,11] is the utilization of an expert grasp stability hierarchy and the experiences of fuzzy or neural tip/slip trial grasps to accumulate a training set for an object category network. For a given class of objects this network ranks a candidate hand approach vector as acceptable for reducing the number of nonoptimal grasps. Objects are grouped based on grasp experience, not visual shape as perceived by humans.

5.2 Problem Approach

The required grasp orientation and pressure are determined by repeating a procedure where, after the hand makes three tentative test grasps, a look-up table stability hierarchy is referenced to determine whether a lift should be attempted with one of these grasps for a given tip or pad contact scenario (Figures 1 and 2), or another three should be tried. Once a grasp is attempted, fuzzy logic tip and slip routines correct for the errors in desired orientation for the object (usually defined as the orientation it was found in). When no tip or slip registers, a rue lift will be attempted and independent trajectory following routines can be initiated. After several such lifting attempts, a neural network learns a best approach angle and hand orientation for a category of objects it has defined from experience. If a set of three test grasps triggers recognition from the neural network, this approach angle and orientation will be implemented and the control algorithm will resume as before with the fuzzy tip and slip algorithms.

In order to overcome the constraints of time and complexity, simplifications are made to the approach which will not, for the time being, compromise the effectiveness of the control, and may in fact improve it. The first step toward simplification is to minimize the number of sensors and actuators, and then secondly, to break the control sequence into smaller manageable routines. Thirdly, an approach or combination of approaches must be chosen which can best handle each task.

Each joint on the finger is assumed to have two parallel pressure sensors (assumed here to be strain gauges), a number adequate to sense a change in orientation perpendicular to the joint grasping surface, as well as to register an absolute value to compare to an acceptable threshold. For the preliminary calculations, placement is first calculated for the first finger and thumb only, leaving the second finger free for balancing of the object.

When one of the two digits contacts the object, the second closes in to make contact. Pressure readings are taken, and if a grasp is to be attempted, the third finger contacts the surface. Tentative grasps can be helpful not only in learning the smallest practical diameter of the object (usually most stable), but also help avoid locations with sudden changes in diameter, such as a handle on a mug or a pocket clip on a pen.

5.3 A Local Grasp Stability Hierarchy

Some grasps are intuitively more stable than others, and should be given a higher priority when several trial grasp options are available. Assuming no open vessels of liquid, a scheme was devised that agrees with one's intuitive perception of which grasps are most stable. The method is demonstrated with an example of four trial grasps shown in Figure 3 (note that the second finger is neglected at first, to be used for extra stability). Figure 4 outlines the membership values assigned to grasps A through D for each criterion. The second joint of grasp B does not touch the object; instead it reaches behind it as if it were a mug handle, for instance.

A grasp hierarchy can be determined once applicable assumptions are made and applied. There are many ways that a grasp can be perceived more stable. The first to be considered is the number of pressure contacts made per finger or thumb. It was also assumed that joints farther away from the wrist can take advantage of supporting objects against the "palm," and are therefore more stable that those objects held by more proximal joints. The two hierarchies in Figures 5 and 6 resulted, and membership function values were assigned, ranging from 1 for the first ranked, and zero for the last, where the finger or thumb makes no contact. The order for the finger ran {1,.9,.8,.7,.3,.2,.1,0}, and the thumb ran {1,.7,.3,0}.

Other criteria also contribute to a more stable grasp. Assuming that the largest positive theta 1 (base grasp) angle corresponds to a clenched fist, and that this would imply a tighter grasp, all candidate grasps were ranked, highest theta one as .8, lowest as .2, and all others equally spaced between. This was done separately for both finger and thumb. Based on the premise that if an object is grasped at two points 180 degrees apart

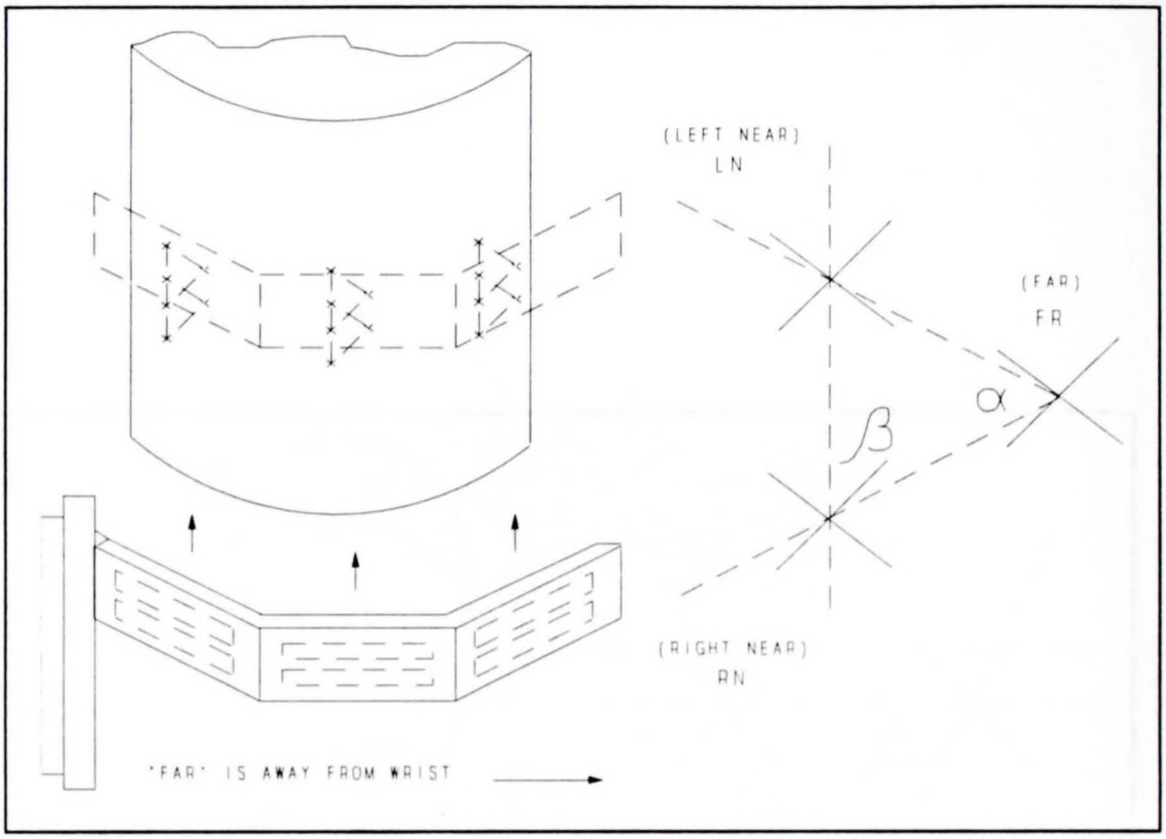

(LEFT NEAR)
LN
(FAR)
FR
α
β
(RIGHT NEAR)
RN
'FAR' IS AWAY FROM WRIST

Figure 1

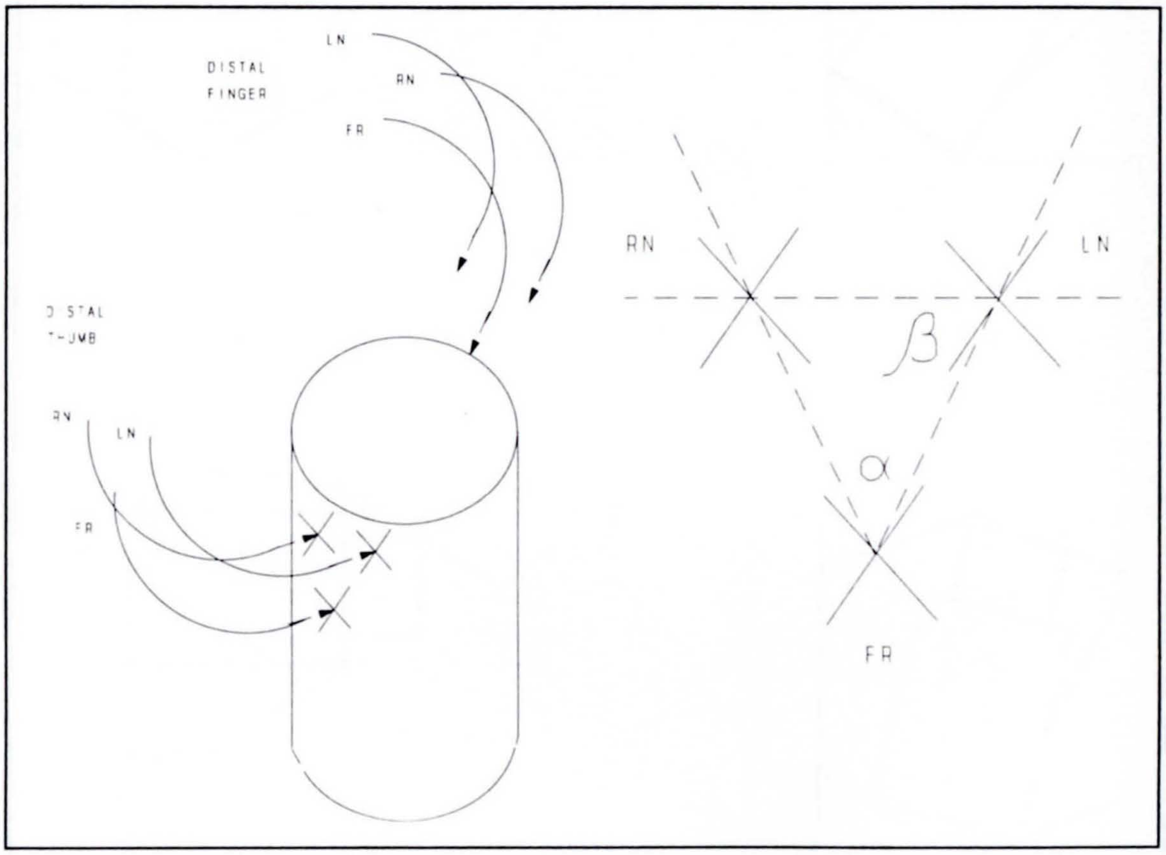
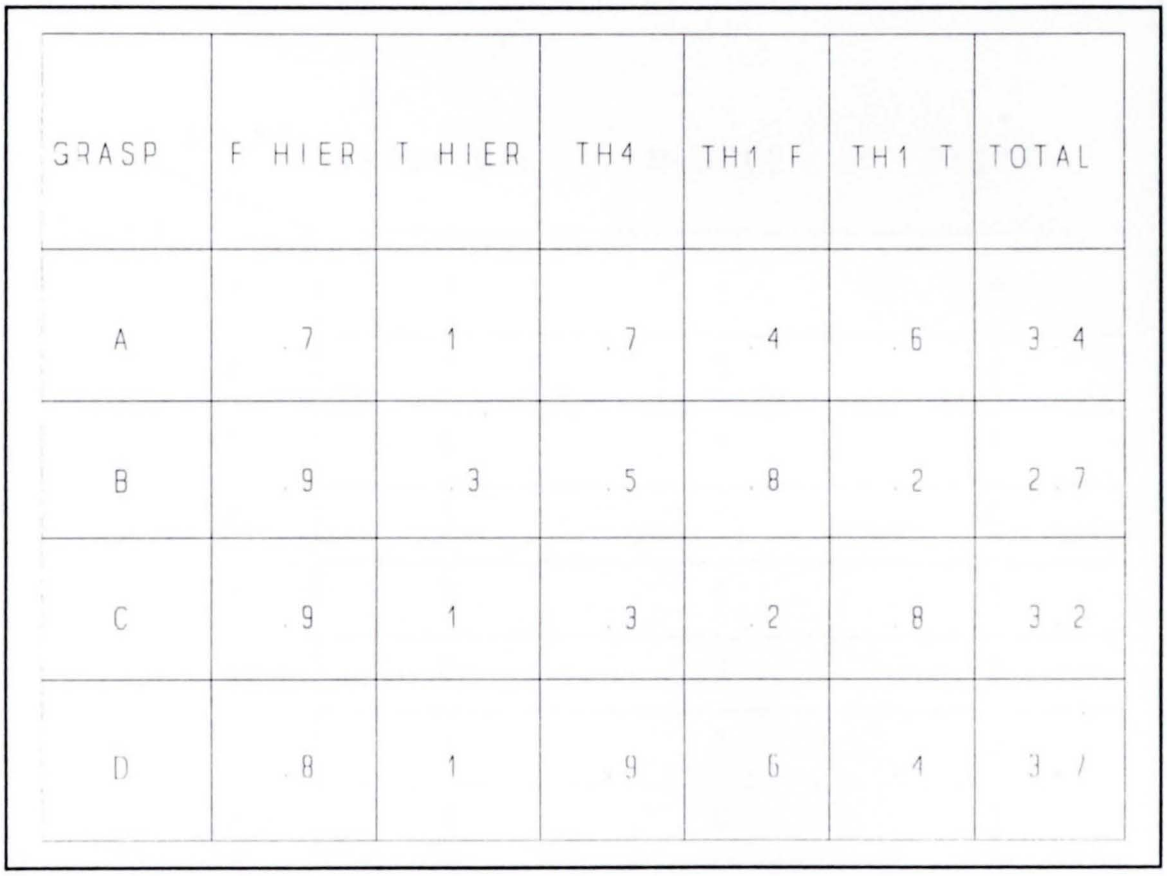
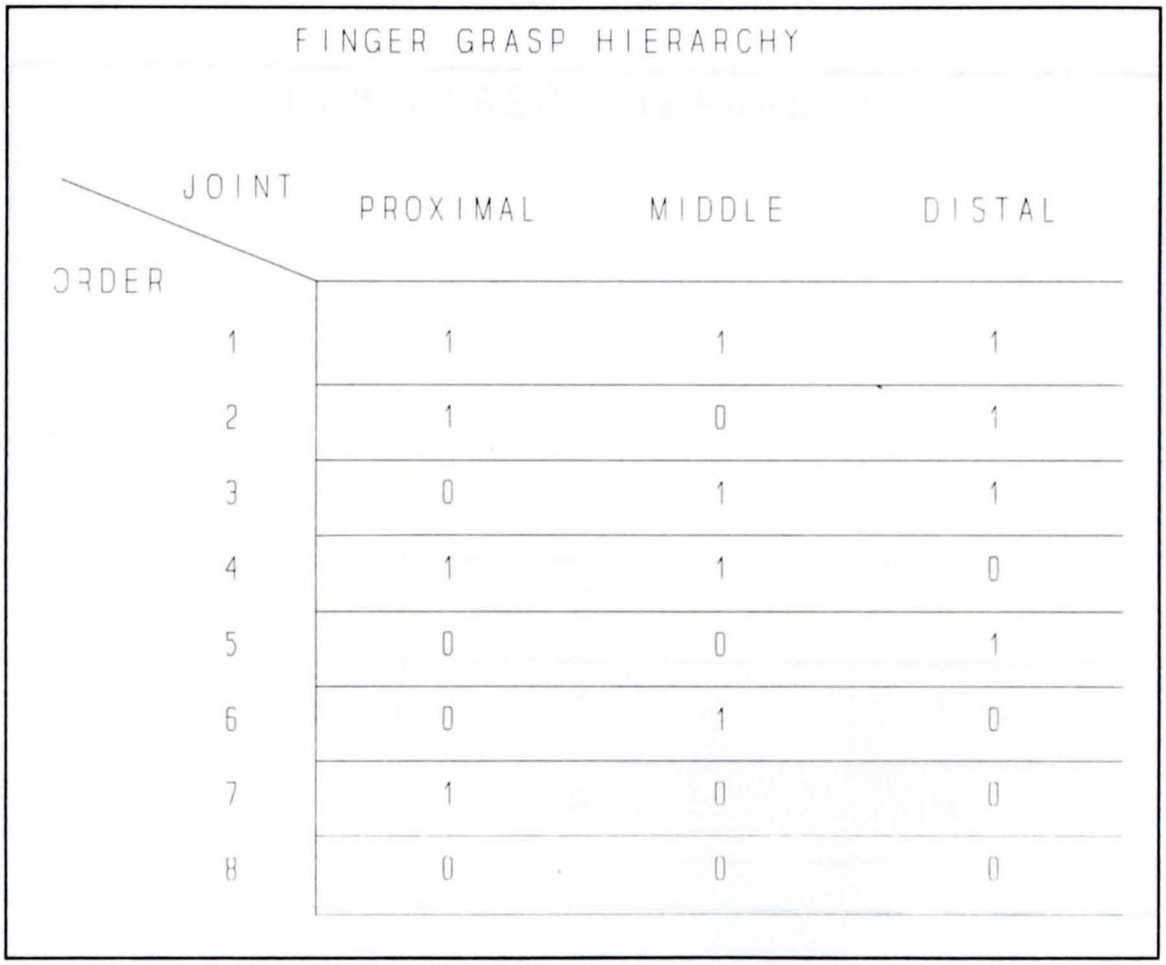

LN
DISTAL
FINGER
RN
FR
DISTAL
THUMB
RN
LN
FR
RN
LN
β
α
FR

Figure 2



GRASP	F HIER	T HIER	TH4	TH1 F	TH1 T	TOTAL
A	.7	1	.7	.4	.6	3.4
B	.9	.3	.5	.8	.2	2.7
C	.9	1	.3	.2	.8	3.2
D	.8	1	.9	.6	.4	3.7

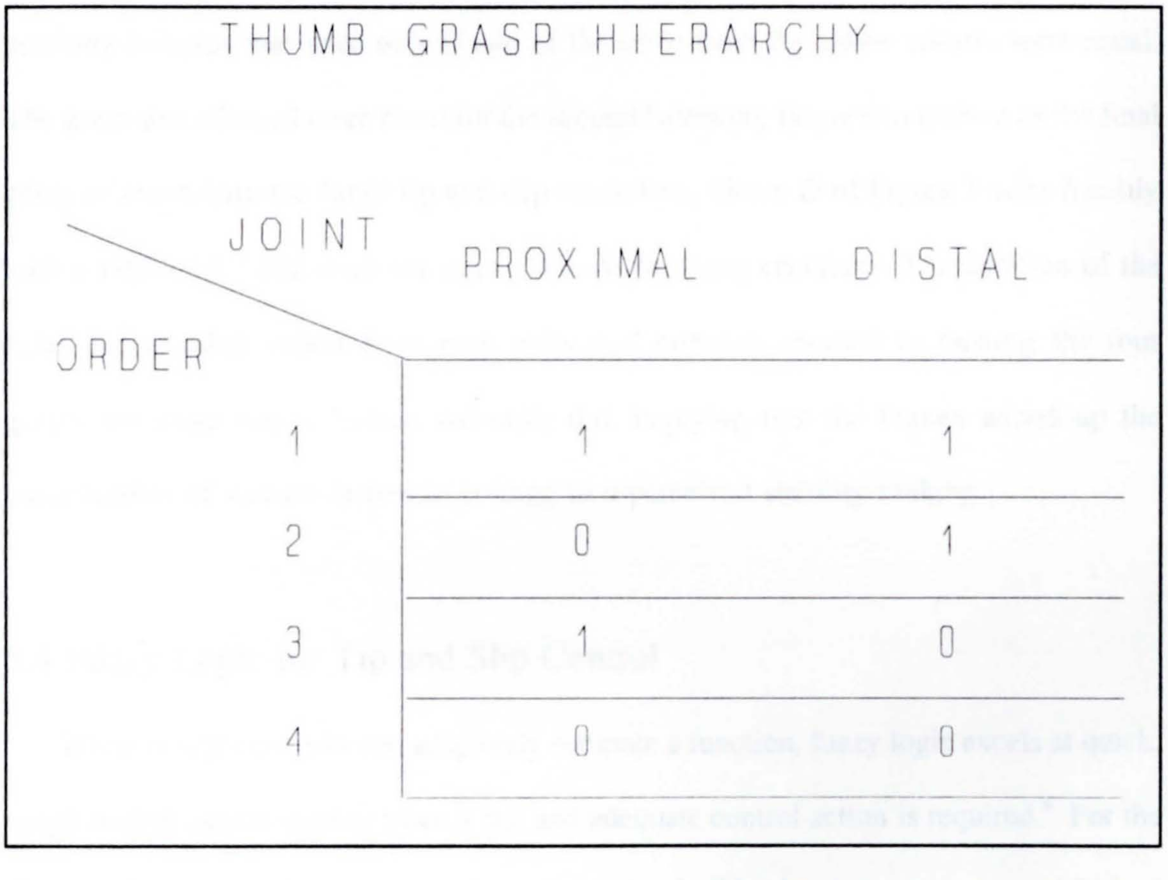
Figure 4



FINGER GRASP HIERARCHY

ORDER \ JOINT	PROXIMAL	MIDDLE	DISTAL
1	1	1	1
2	1	0	1
3	0	1	1
4	1	1	0
5	0	0	1
6	0	1	0
7	1	0	0
8	0	0	0

Figure 5



THUMB GRASP HIERARCHY

ORDER \ JOINT	PROXIMAL	DISTAL
1	1	1
2	0	1
3	1	0
4	0	0

Figure 6

it is most stable, the highest rank of .9 was assigned to finger and thumb combinations that most closely satisfied the equation:

$$(\theta_4)_{finger} = -(\theta_4)_{thumb} \qquad (1)$$

and the lowest rank was assigned .3. All others were spaced equally as before. A tie breaking criterion was used only if two of the sums from the above criteria were equal. The grasp that allowed more room for the second balancing finger was chosen as the final grasp to move onto the fuzzy tip and slip controller. Grasp D of Figure 5 wins handily with a total of 3.7 and does not require the tie breaking criterion. The addition of the fuzzy membership values from each individual criterion resulted in ranking the four grasps the same way a human volunteer did, implying that the human added up the contributions of various factors in coming to a perceived stability ranking.

5.4 Fuzzy Logic for Tip and Slip Control

While neural networks can adaptively estimate a function, fuzzy logic excels at quick, rough human approximation when a fast and adequate control action is required.[9] For the simplest fuzzy controller, common sense is used to build rules that, given normal "crisp" sensory values (i.e., position, velocity, torque, or pressure), percentage memberships in various categories are determined. A logical process (often max-min decomposition) is used to choose the dominant category, and a resulting control effort is determined through defuzzification to a crisp value. Because fuzzy logic obtains its commands from additions and multiplications of non-zero set membership values, classical control mathematics violations are avoided, and time is optimized.

In order to adjust the grasp for object tip and slip employing fuzzy set theory, a three step algorithm is followed. First, it is assumed that excessive pressure on one member of a pair of pressure sensors for a finger in Figure 7 signals a possible tipping of the object. The bottom member of each pair is subtracted from the top, and the values are entered along the horizontal axis of the membership function in Figure 8. A membership value is found on the vertical axis for every category that is nonzero at that horizontal value. Category values are found for each sensor pair that exceeds a minimum tolerance, ε. This tolerance ignores pairs that do not register a significant pressure as well as pairs that are practically identical and do not need adjustment.

Sensor input categories must now be defined, and possible controller responses determined. The minimum membership values for each category, large negative (LN), small negative (SN), near zero (NZ1), small positive (SP), and large positive (LP) are found, and the category that possesses the maximum of these surviving minimums is the category chosen for control. In the cases of LN, LP, and NZ1, the PUMA arm that supports the hand will be given the command to move the finger up or down the object (assuming a loosing and subsequent tightening of the finger some small amount). If the winning category is SN or LP, the fingers themselves can adjust their yaw angles to compensate. The actual adjustment command is found by entering the associated membership into the vertical axis of a function such as Figure 9, retrieving the desired command value at the horizontal. Humans do not react symmetrically in each direction,

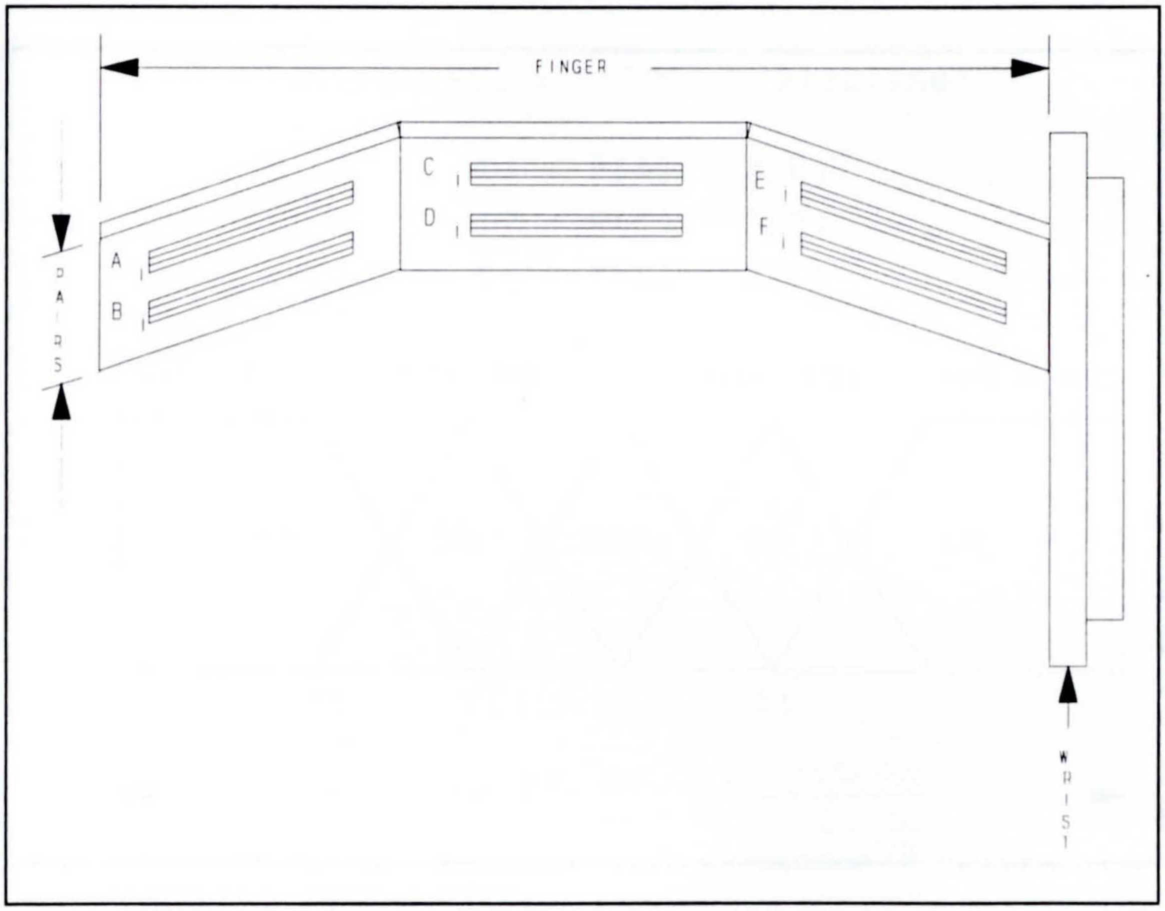


Figure 7

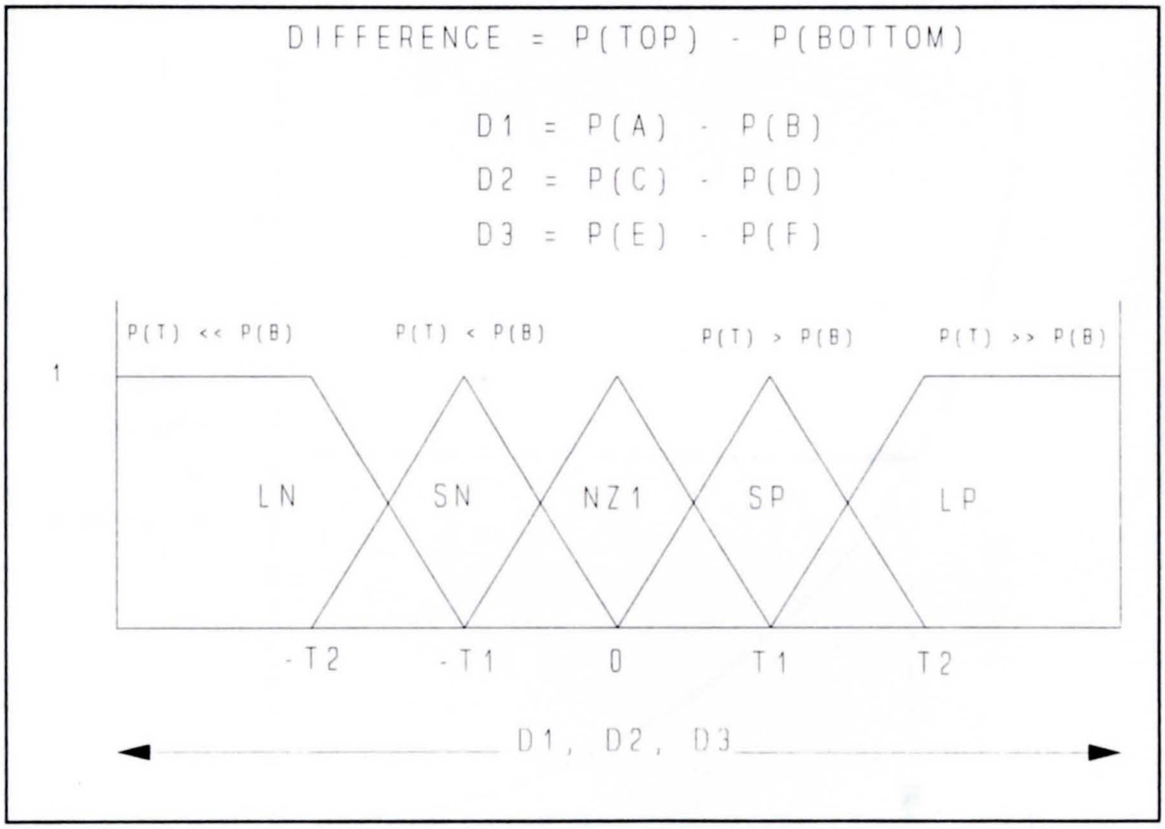

DIFFERENCE = P(TOP) - P(BOTTOM)
D1 = P(A) - P(B)
D2 = P(C) - P(D)
D3 = P(E) - P(F)
P(T) << P(B)
P(T) < P(B)
P(T) > P(B)
P(T) >> P(B)
1
LN
SN
NZ1
SP
LP
-T2
-T1
0
T1
T2
D1, D2, D3

Figure 8

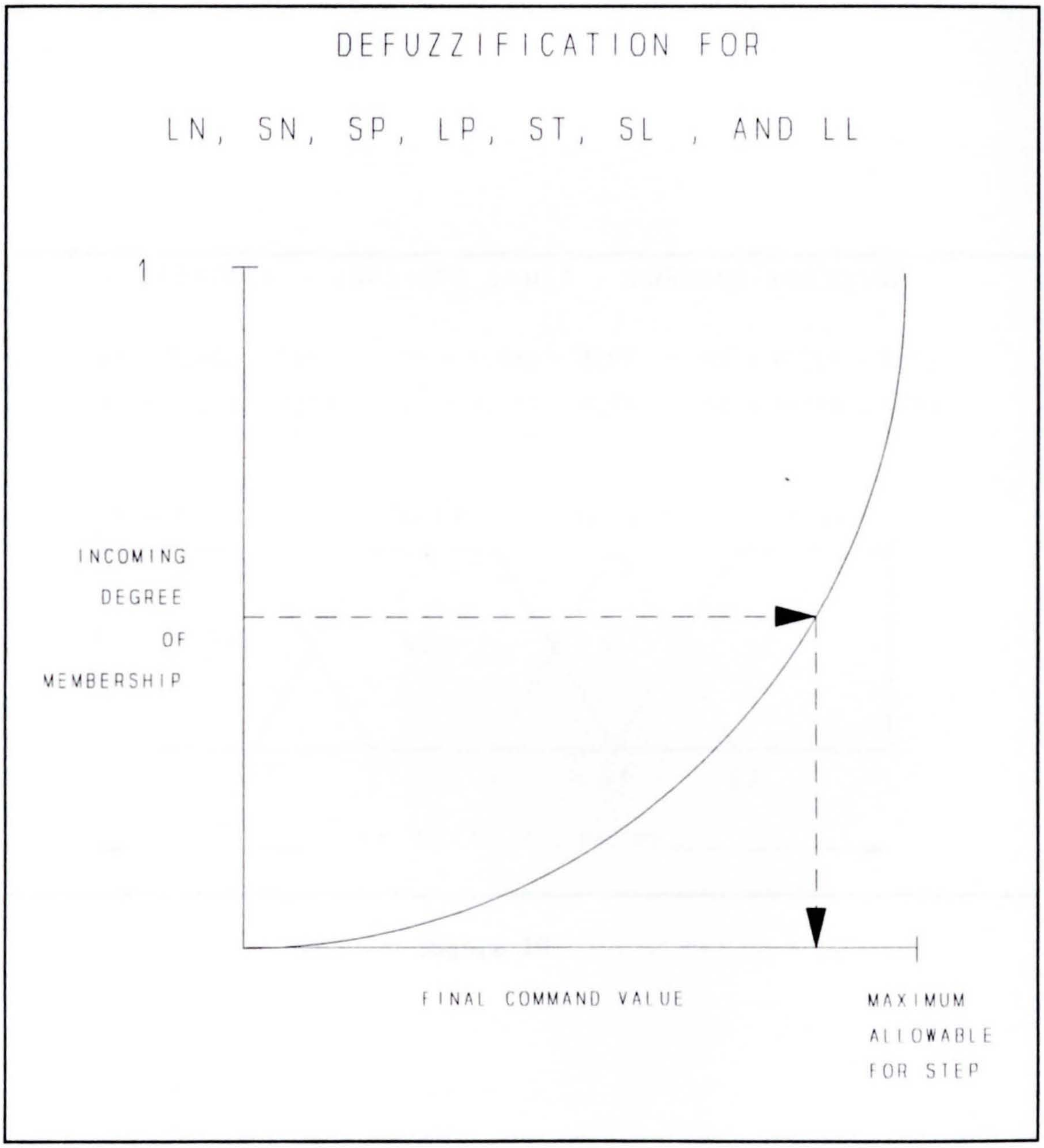

DEFUZZIFICATION FOR
LN, SN, SP, LP, ST, SL , AND LL
1
INCOMING DEGREE OF MEMBERSHIP
FINAL COMMAND VALUE
MAXIMUM ALLOWABLE FOR STEP

Figure 9

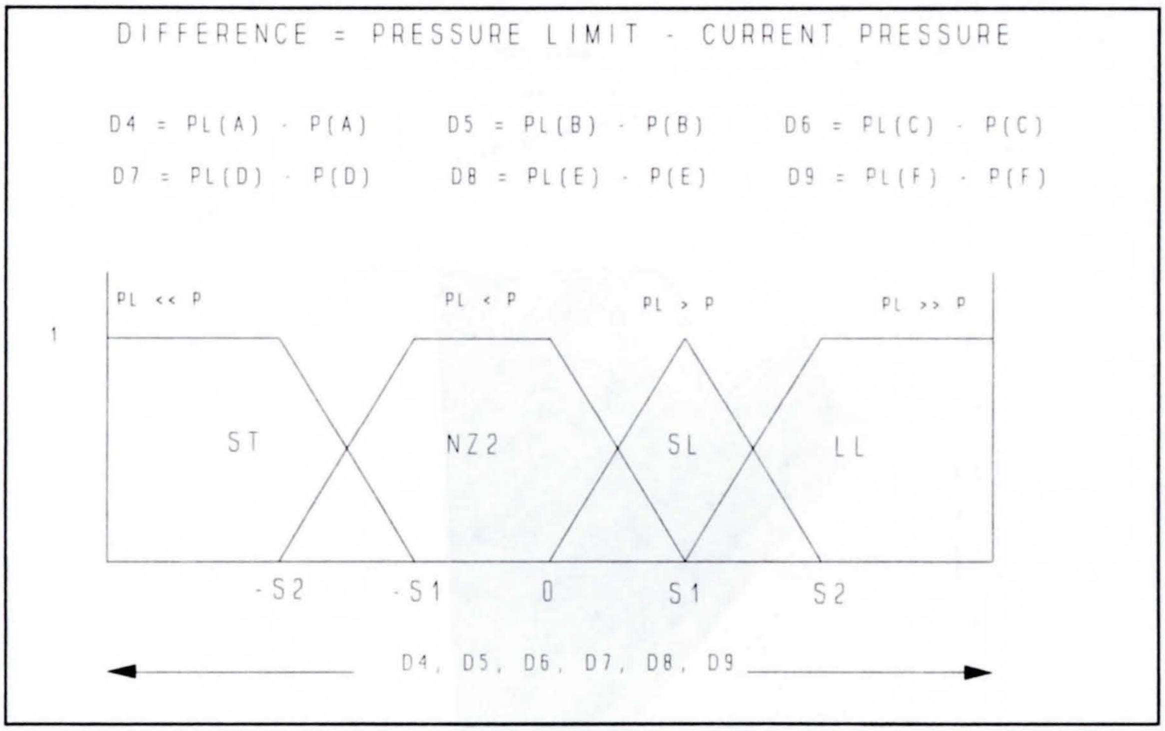

DIFFERENCE = PRESSURE LIMIT - CURRENT PRESSURE
D4 = PL(A) - P(A)
D5 = PL(B) - P(B)
D6 = PL(C) - P(C)
D7 = PL(D) - P(D)
D8 = PL(E) - P(E)
D9 = PL(F) - P(F)
PL << P
PL < P
PL > P
PL >> P
1
ST
NZ2
SL
LL
-S2
-S1
0
S1
S2
D4, D5, D6, D7, D8, D9

Figure 10

so in order to avoid a limit cycle, avoid symmetry in the fuzzy controller command function.

The second major focus in the algorithm, the slip problem, is very similar to that for tip, but the incoming pressure information must be interpreted differently. Each pressure sensor is compared to a suggested pressure limit, and these limits may vary due to individual sensor characteristics or to expected pressures for a perceived familiar shape. It is assumed that a little too tight of a grasp is better than a little too loose, so no adjustment is made for anywhere from just right to a little too tight. Much too tight, however, could damage the finger or grasped object, so the finger is carefully loosened. Loose grasps are tightened to approach the desired limit. In a manner similar to that for tip, each individual pressure is subtracted from its limit and the values are entered onto the horizontal axis of Figure 10.

For slip, the choice of control signal must be considered carefully in case the shape of the grasped object produces confusing inputs. This time, instead of employing a maximum of the minimums to decide on a prevailing control effort, a priority hierarchy is followed. If any membership value for small tight (ST) is equal to 1, all other values are overruled and the grasp is loosened. Otherwise, if the minimum for ST is greater than .1, that value is entered into a function similar to Figure 9, and that command is used for adjustment. Otherwise, the .1 criterion is applied to near zero (NZ2) if true, or small loose (SL) if false. Finally, if no previous category has a minimum greater than .1, the appropriate adjustment is found by inserting the minimum for large loose (LL).

An example of a confusing slip situation is illustrated in Figure 11. Here the middle finger pair has exceeded the suggested pressure limit, ostensibly crushing the object, while a small offshoot of the object barely makes contact with one of the two sensors proximal to the wrist. In this case the two middle sensors would probably have overruled the proximal sensor nonetheless, but an object with many projections could draw attention away from a critical stress situation.

If an adjustment for tip or slip is not necessary, the third phase, lifting of the object, may be attempted. First, the algorithm checks to see if the winning categories for the two prior steps were NZ1 and NZ2. If so, an attempt is made to lift or continue lifting the object before repeating steps 1 and 2. If large negative (LN) or large positive (LP) are resulting commands for tip after the object has been lifted, an attempt should be made to set the object down before readjusting, and the estimated neural grasp should be recalculated before lifting the object again.

Pseudocode for the Tip and Slip Algorithms follows:

FUZZY TIP CONTROLLER:

```
Program Tip_Controller
{
  While (Press_Differences > Stopping Criterion)
    Receive Press_Values from Hand
    For each active Finger_Joint within an
                       approach vector region
```

```
        Find Press_Differences
      Endfor
      For each Membership_Function
        Call Fuzzify subroutine
      Endfor
      Winning_Value = Max(all Minvals)
      Winning_Mem_Function = Membership_Function(Winning_Value)
      For each Lookup_Defuz
        If assigned to Winning_Mem_Function
          Output_Tip = Lookup_Defuz(Winning_Mem_Value)
        Endif
      Endfor
      If Press_Differences in opposing approach vector
                                                   region > ε
        Output_Tip = Winner of comparison of Output_Tips
      Endif
      Call Slip_Controller subroutine
    Endwhile
    Attempt_Hand_Lift
}
```

FUZZIFY SUBROUTINE:

```
Subroutine Fuzzify
{
  Initialize Minval(Membership_Function) as a number > 1
  For each Press_Difference
    If Press_Difference is supported by Membership_Function
      Calculate Membership_Value ( (0,1] )
    Endif
    If Membership_Value < Minval(Membership_Function)
      Minval(Membership_Function) = Membership_Value
    Endif
  Endor
}
```

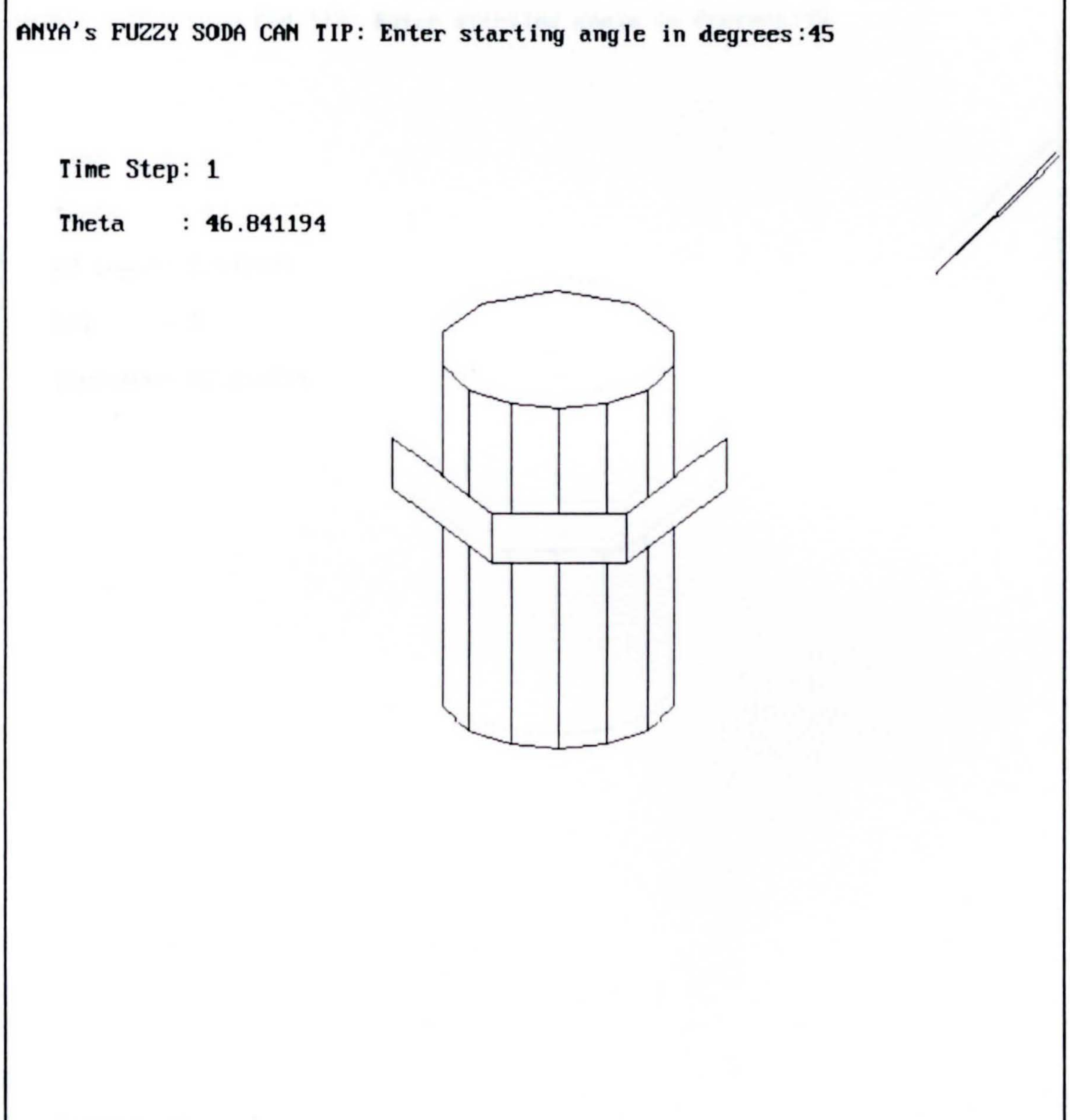
FUZZY SLIP CONTROLLER:

```
Subroutine Slip_Controller
{
  Receive Press_Values from Hand
  For each activated Finger_Joint within an
                        approach vector region
    If Press_Values > Press_Tolerance
      Loosen Hand_Grasp
      Goto Tip_Controller
    Endif
  Endfor
  For each Membership_Function in Slip_Hierarchy_Order
    Call Fuzzify subroutine
    If Minval(Membership_Function) > .1
      Winning_Value - Minval(Member_Function)
    Winning_Mem_Function = Membership_Function
    Break out of For Loop
   Endif
  Endfor
  If Press_Values in opposing approach vector region > ε
   Output_Tighten = Winner of comparison of Output_Tightens
  Endif
}
```

5.5 A Fuzzy Logic Tipping Controller

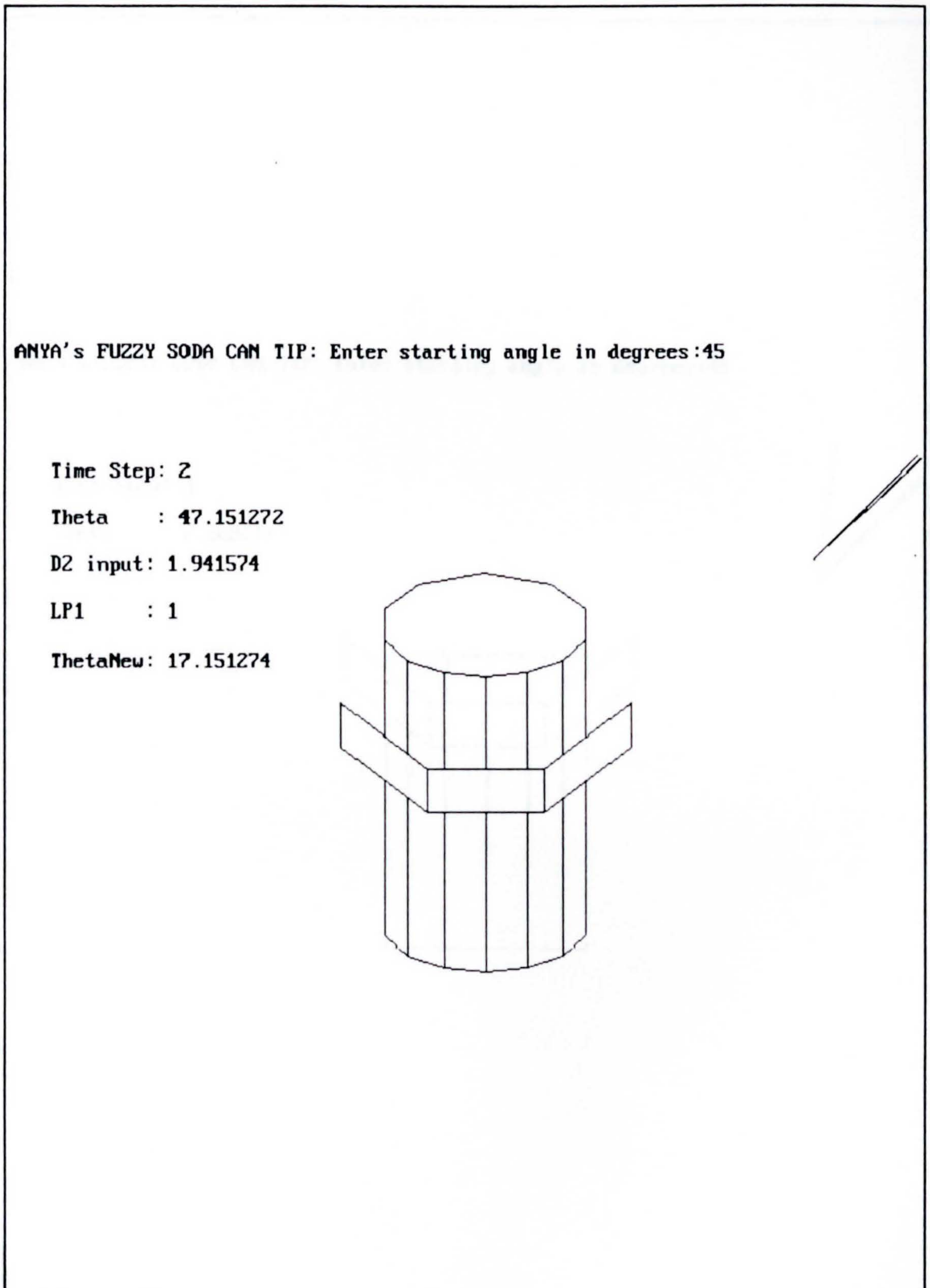
A simplified tipping case was demonstrated by a simulation written in C, which will be extended to three dimensions. Some assumptions have been made to approximate reality. First, the motion of the soda can is modelled as an inverted pendulum with a stable equilibrium state at topmost vertical (within ± 1.7 degrees). Friction at the soda can's bottom pivot is assumed strong enough to ignore slipping on the supporting surface.

Tipping is confined to movement into or out of the screen, and only one finger is being used to counter the tipping. The thumb is fixed in the location estimated by the neural net, which is a useful assumption if the thumb is within a mug handle, for instance. The finger adjusts with a vertical up or down movement, pushing the can backward by a number of degrees in proportion to its vertical movement. The finger adjustments are slow to avoid drastic overshoot and to allow an overworked controller to catch up. The can is allowed to continue in its motion before the next adjustment.



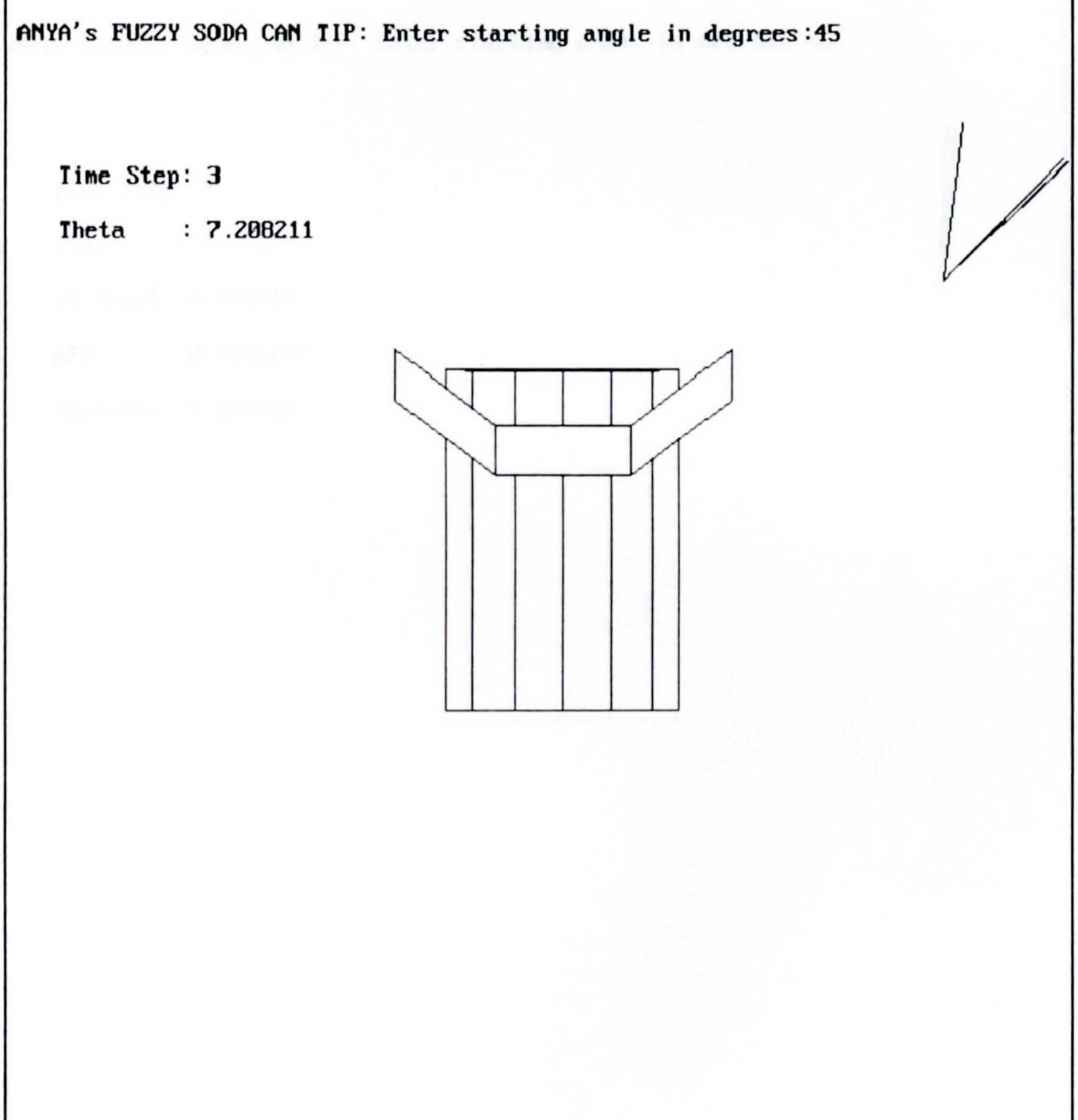
```
ANYA's FUZZY SODA CAN TIP: Enter starting angle in degrees:45

    Time Step: 1

    Theta     : 46.841194
```

Figure 12



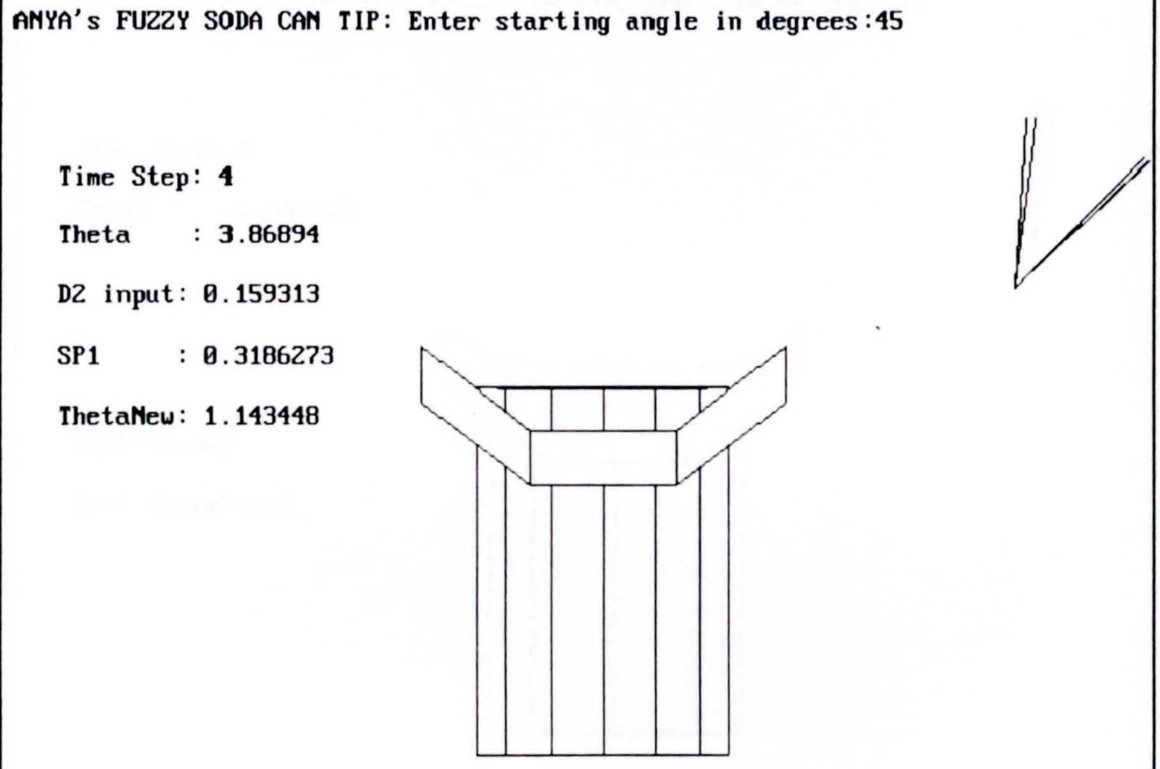
```
ANYA's FUZZY SODA CAN TIP: Enter starting angle in degrees:45

   Time Step: 2

   Theta    : 47.151272

   D2 input: 1.941574

   LP1      : 1

   ThetaNew: 17.151274
```

Figure 13



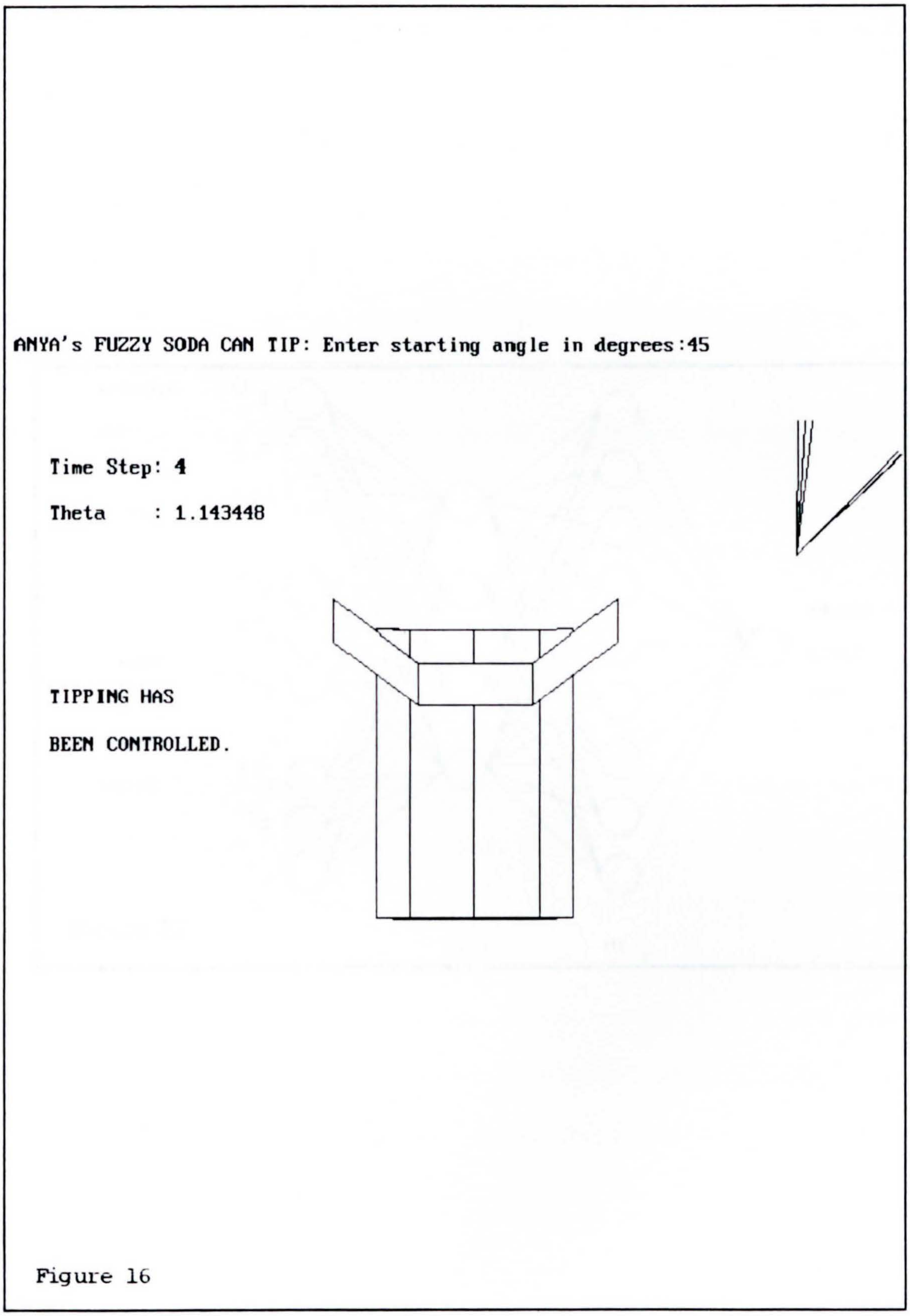
```
ANYA's FUZZY SODA CAN TIP: Enter starting angle in degrees:45

    Time Step: 3

    Theta      : 7.208211
```

Figure 14



```
ANYA's FUZZY SODA CAN TIP: Enter starting angle in degrees:45

   Time Step: 4

   Theta    : 3.86894

   D2 input: 0.159313

   SP1      : 0.3186273

   ThetaNew: 1.143448
```

Figure 15



```
ANYA's FUZZY SODA CAN TIP: Enter starting angle in degrees:45

   Time Step: 4

   Theta     : 1.143448


   TIPPING HAS

   BEEN CONTROLLED.
```

Figure 16

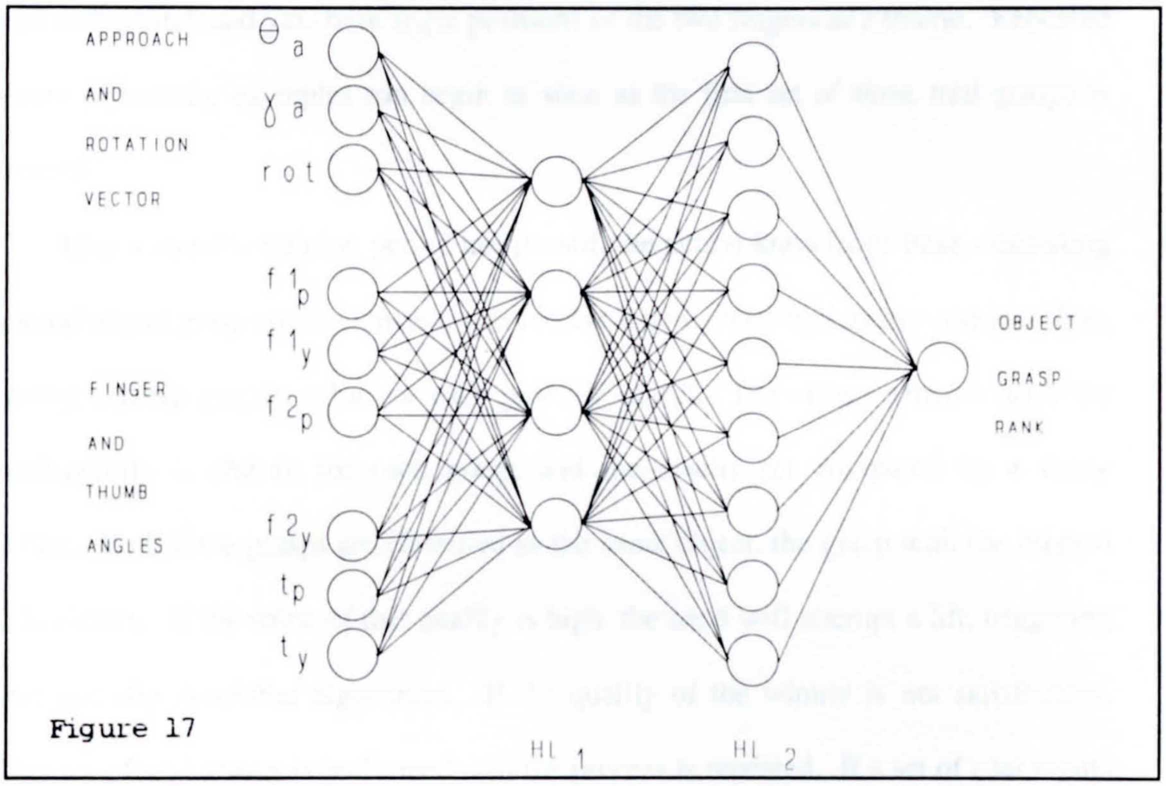

APPROACH AND ROTATION VECTOR
θ_a
γ_a
rot
$f1_p$
$f1_y$
$f2_p$
$f2_y$
t_p
t_y
FINGER AND THUMB ANGLES
HL 1
HL 2
OBJECT GRASP RANK

Figure 17

Pseudocode for the Neural Network Object Category Estimator Follows:

NEURAL NETWORK OBJECT CATEGORY ESTIMATOR:

Run Tip_Controller for several trials on some training objects.

Then:

```
Program Train_Estimator
{
  For each training object
    Receive Num_of_Trials, Approach_Theta, Approach_Phi,
                    Hand_Rotation, $fing1\_q_{pitch}$, $fing1\_q_{yaw}$,
                     $thumb\_q_{pitch}$, $thumb\_q_{yaw}$, $fing2\_q_{pitch}$, and
                    $fing2\_q_{yaw}$ from Tip Controller
    Scale Num_of _Trials to fit a range from 0 to MaxTrials
    Train a Category_NNet to match scaled Num_of_Trials for
                    object
    Separately store values of Approach_Theta,
                    Approach_Phi, and Hand_Rotation which
                    result in lowest Num_of_Trials value.

  Endfor
```

efficient communication between its diagnostic nodes and the originators of joint commands. The preliminary results look promising for a reasonable accumulation of information in a form that useful to this level of the hierarchy as well as others.

References

[1]Hu, Huosheng, Michael Brady, and Penelope Probert, "Navigation and Control of a Mobile Robot Among Moving Obstacles," **Proceedings of the 30th Conference on Decision and Control**, Brighton, England, December 1991.

[2]Gongliang Guo, William A. Gruver, and Qixian Zhang, "Optimal Grasps for planar Multifingered Robotic Hands," **IEEE Transactions on Systems, Man, and Cybernetics**, Vol. 22, No. 1, January/February 1992.

[3]Nguyen, Thang N., and Harry E. Stephanou, "A Continuous Model of Robot Hand Preshaping," **IEEE International Conference on Systems, Man, and Cybernetics**, New York: Institute of Electrical and Electronic Engineers, Inc., pp. 798-803, 1989.

[4]Nasrabadi, Nasser M., and Wei Li, "Object Recognition by a Hopfield Neural Network," **IEEE Transactions on Systems , Man and Cybernetics**, Vol. 21, No. 6, November/December 1991.

[5]Takagi, Toshiyuki, and Shohachiro Nakanishi, "Pattern Recognition Based on the Extraction of Features by Neural Networks and Fuzzy Set Theory," **Part VI, Fuzzy and Neuro, Fuzzy Engineering toward Human Friendly Systems**, IFES 1991.

[6]Eng, Victor J., "Modal Segments for a Peg Insertion Task," **IEEE Robotics: Challenges in Motion Control, Planning, and Vision**, The 29th Videoconference Seminars via Satellite, Thursday, April 6, 1989.

[7]Scharf, E.M., and N.J. Mandic, "The Application of a Fuzzy Controller to the Control of a Multi-Degree-of-Freedom Robot Arm," **Industrial Applications of Fuzzy Control**, M. Sugeno, editor, North-Holland: Elsevier Science Publishers B.V., 1985.

[8]Wang, H., T.T. Lee, and W.A. Gruver, "A Neuromorphic Controller for a Three-Link Biped Robot," **IEEE Transactions on Systems, Man, and Cybernetics**, Vol. 22, No. 1, January/February 1992.

[9]Zadeh, L.A., "Fuzzy Sets," **Information and Control**, No. 8, 1965, pp. 338-353.

[10]Tascillo, A., N. Bourbakis, and J. Crisman, "Intelligent Control of a Robotic Hand with Neural Nets and Fuzzy Sets," **Proceedings IEEE Symposium on Intelligent Control**, Chicago, Illinois, August 1993, pp. 232-237.

[11]Tascillo, A. and N. Bourbakis, "A Neurofuzzy Grasping of a Robot Hand," **IEEE Transactions on Systems, Man, and Cybernetics**, to appear 1995.

CHAPTER 6:

MULTIPLE ARM AND OBJECT PLANNING

6.1 Problem Scenario

This chapter presents a planning methodology based on Stochastic Petri Nets (SPN) and Neural nets for coordination of multiple robotic arms working a space with constrained placement. The SPN planning method generates a global plan based on the states of the elements of the Universe of Discourse. The plan includes all the possible conflict-free planning paths to achieve the goals under constraints, such as specific locations on which objects have to be placed, order of placement, etc. An associated neural network is used to search the vectors of markings generated by the SPN reachability graph for the appropriate selection of plans. Moreover, it preserves all the interesting features of the SPN model, such as synchronization, parallelism, concurrency and timing of events. The coordination of two robotic arms is used as an illustrative example for the proposed planning method, in a UD space where the location of objects placement are restricted.

There are a variety of planning methodologies developed in the last twenty years.[1-13] Most of these methodologies (NOAH, NONLIN, DEVISER, SIPE, TWEAK, etc.) generate a partially ordered plan network for the achievement of conjunctive goals. More specifically, these methods are based on state oriented planning, where each plan is

CHAPTER 6:
MULTIPLE ARM AND OBJECT PLANNING

6.1 Problem Scenario

This chapter presents a planning methodology based on Stochastic Petri Nets (SPN) and Neural nets for coordination of multiple robotic arms working a space with constrained placement. The SPN planning method generates a global plan based on the states of the elements of the Universe of Discourse. The plan includes all the possible conflict-free planning paths to achieve the goals under constraints, such as specific locations on which objects have to be placed, order of placement, etc. An associated neural network is used to search the vectors of markings generated by the SPN reachability graph for the appropriate selection of plans. Moreover, it preserves all the interesting features of the SPN model, such as synchronization, parallelism, concurrency and timing of events. The coordination of two robotic arms is used as an illustrative example for the proposed planning method, in a UD space where the location of objects placement are restricted.

There are a variety of planning methodologies developed in the last twenty years.[1-13] Most of these methodologies (NOAH, NONLIN, DEVISER, SIPE, TWEAK, etc.) generate a partially ordered plan network for the achievement of conjunctive goals. More specifically, these methods are based on state oriented planning, where each plan is constructed by generating a subplan for each goal state, while the detection and resolution of conflicts among these subplans take place in the state space. In addition, a well defined planning methodology, called RPP[8], has been proposed for the resolution of parallel plans. This method considers resources as active elements in constructing plans and generates conflict-free subplans by controlling the flow of a particular resource, while it synthesizes a complete plan in cooperation with the other subplans. The RPP methodology constructs conflict free subplans prior to the expansion of the current plan network. Another methodology uses heuristic search on a Petri-net framework and a metric space for efficient planning[9].

Most of the methods discussed above lack in timing and stochastic synchronization of the events (actions) which take place during the execution of a plan. Moreover, most of them cannot efficiently handle cases where the number of elements increases significantly.

In this paper, a planning methodology based on the combination of stochastic Petri-nets and neural nets is proposed. More specifically, the plan network is expressed as an SPN network where all the states of the elements and the actions of the universe of discourse are expressed as Petri-net places P and transitions T respectively, in various levels of abstraction. A neural network model searches the SPN framework for the appropriate selection of plans. Decomposition and synthesis of subplans can be easily achieved on the SPN model. A neural net model is used to search the state space in order to achieve a desirable path. An illustrative example is provided by using the SPN network methodology, for the coordination of two robotic arms in a space with constraints such as specific locations for placement of objects.

The author's contribution[18,19] is the use of an adaptive resonance theory neural network to direct the robot hands toward a minimum time solution given possibilities defined by a stochastic petri net. A network trained to achieve a desired configuration of blocks can achieve this goal given starting configurations it has never seen. This paper is organized into five sections. Section 6.2 presents the important notations and definitions. Section 6.3 deals with the SPN modeling of the planning methodology. Section 6.4 presents the results of the illustrative example and section 6.5 summarizes the overall work.

6.2 Notations and Definitions

Notation 1: The set (UD) of all the elements Ei, $i \varepsilon Z$, and the knowledge related with them for a particular domain:

UD = {Ei/ Ei is an element with properties {F(Ei)}, and {R(Ei,Ej)} relationships among elements}

Notation 2: A state S(k,t), with $k \varepsilon Z$ and t=time, is a representation of the set UD (or a subset of it) at a given time t after an action A.

Notation 3: An action Am, $m \varepsilon Z$, represents the mapping of UD (or a subset of it) from the state S(i,t) into state S(j,t'):

$$Am : \Sigma \times S \dashrightarrow S$$

where Σ ={Am/ Am is a primitive action performed on Ei}, and S = {S(k,t)/ S(k,t) is any state of UD}

A plan would be considered as a sequence of actions performed on the elements of UD at a state for the achievement of the goal state. More specifically,

Definition 1: A plan PLi = {{Am},{Ei},{S(i,t)}} is a tri-tuple, where {S(i,t) ε S} is a sequence of states achieved by performing a sequence of actions {Am ε Σ, m ε Z} on the set of elements at {Ei ε UD, i ε Z} starting from a state S(j,t0) for the accomplishment of the goal states {S(k,tn),n ε Z}.

$$PLi : UD[Ex\{S(k,t)\}] \xrightarrow[\{Am\}]{} UD[Ey\{S(j,t')\}]$$

Notation 4: A plan is successful [PLi] if and only if it can achieve the goal state.

Notation 5: The set of all the successful plans represents the direct graph (or network) G. It is the synthesis of all [PLi], i ε Z :

$$G = \underset{i}{\$} \{[PLi]\}$$

Notation 6: The set G can also expressed as the union of all transformations of the elements of UD which participate in the plan PLi(En):

$$G = \underset{n}{U} G\{PLi(En)\}.$$

6.3 The SPN Planning Methodology

In this section the SPN planning methodology is presented.

6.3.1 Why Stochastic Petri-Nets

There are a variety of methodologies used for planning, such as formal languages, directed graphs, classical mathematical models, queuing models, Petri-nets, etc. In this chapter, a modified version of the Petri-net will be used as the modeling framework of an SPN Planning strategy for two robot hands. The major reasons for the use of the Petri-net are[9,13-17]:

* SPN is an efficient modeling tool for the functional description and analysis of complex systems,
* SPN is able to describe simultaneously concurrency, parallelism and synchronization of events that take place in a complex system, especially when other methodologies lack adequate results,
* SPN can be used as a modeling tool for hierarchical and abstracted (top-down or bottom-up) processes,
* SPN provides timing during the execution of various events,
* SPN presents compatibility with neural nets, and
* SPN is an efficient interface for control and communication.

Notation: The Petri-net used in our case is a Stochastic (SPN) one with time capabilities.

Notation: RT(Mk,NO,FS) represents the reachability tree of SPN.

Notation: RT(Mr) is the set of all markings which are reachable from Mr, $r \varepsilon Z$.

Notation: NO represents the number of moving objects.

Notation: FS represents the number of open corridors.

Note that the SPN used here is a k-bounded one. Also, the association of random time with the transitions will be used. In particular, the SPN transitions will be classified into categories: 1) immediate transitions (which fire in "zero" time once they are enabled) and 2) the timed transitions (which fire after a random, exponentially distributed, enabling time). This assumption will reduce the complexity of SPN when it is needed.

6.3.2 The SPN Model

Definition 2: The structure of the SPN planning model is based on a stochastic Petri-net model. Thus, its formal definition is an 11-tuple :

SPN(P) = {P,T,I,O,M,L,X,MC,Q,R,F}

where

* P ={P1,P2,P3,...,Pn} is a finite set of places. Each place Pj represents a particular state of an element Ei;
* T ={T1,T2,T3,...,Tm} is a finite set of transitions. A transition Ti, iεZ, represents an action performed on a set of elements at a state S(k,t);
* Ii C (PxT) is the input function and
* Oj C (TxP) is the output function;
* Mi = {mi1,mi2,mi3,...,min}, ijεZ, is the vector of marking (tokens) mij, which represent the status of the places; for i=0 m0j, j=0,1,...,n denotes the number of tokens in place Pj in the initial marking M0.
* Q = {t1,t2,t3,...,tm} is the vector of time values related with the time required by an action to be performed.

* R = {r1,r2,r3,...,rk} is the set of the relationships among elements;
* F = {f1,f2,f3,...,fn} is the set of the properties of the elements.
* MC = { a1,a2,...,ak }. k∈Z, is the alphabet of communication,
* L = { l1,l2,...lm } is the set of possibly marking-dependent firing rates associated with the Petri-net transitions, and
* X = { x1,x2,...,xm } is the set of delays associated with the Petri-net transitions.

Notation: The Petri-net used in our case is a Stochastic (SPN) one with time capabilities.

Notation: RT(Mk) represents the reachability tree of SPN.

Notation: RT(Mr) is the set of all marking which are reachable from Mr, rεZ.

Notation: The SPN used here is a k-bounded one.

Notation: The association of random time with the transitions will be used:

1) immediate transitions (which fire in "zero" time once they are enabled)

2) the timed transitions (which fire after a random, exponentially distributed, enabling time).

Notation: A marking Mki ε RT represents a node of the reachability tree.

Definition : An SPN plan is safe if the number of tokens in each place is less or equal to 1, for all markings in RT(Mi).

Definition : The length v of an SPN path is the number of markings Mk, k ε Z, which compose the path.

Definition : The length ls of an SPN plan is the number of transitions required, from an initial marking Mk0, to reach the goal markings Mk ε SPNPL.

Definition : An SPN path is empty if v=0 and unique if v=1.

Definition : An SPN plan is empty if ls=0 and unique if the number of paths Lp, which compose the plan is Lp=1.

Definition: An SPNPLi plan is closed if there is no transition from any state of SPNPLi to any state of ^SPNPLi, where ^SPNPLi, i ε Z represents the complement plan.

Proposition: Two SPN plans are equivalent, SPNPLi = SPNPLj if lsi = lsj and have the same cost.

Proposition: The synthesis (@) of two paths requires the union of the marking sequences.

Proposition: The synthesis of two SPN plans requires the union of the paths and the appropriate adjustment of the transitions associated with the markings.

Proposition : SPNPHi @ SPNPHj ? SPNPHj @ SPNPHi; and SPNPLi @ SPNPLj ? SPNPLj @ SPNPLi.

6.3.4 SPN Planning in UD Using Two Robotic Arms

Let us consider the set UD = {a,b,c, {on,table, available, clear}, {above,left,right}, {L1,L2,L3} } as shown in Figure 1, where Li, i=1,2,3, represents a particular location on the table. Also the set of actions Σ = {start(sr),stop(sp), move(mv),grab(gr),pick-up(pu),release(re),stack(st), unstack(us),put-down(pd),wait(w)}. Then, the plan-network G for two successful plans is shown in Figure 2. The decomposition of G into three subplans G{P(a)}, G{P(b)}, G{P(c)} is shown in Figure 3, under the condition of parallel execution by three independent units[8]. In the case where only two units (robot hands) are available, then the subplans are presented in Figure 4. It is understandable that in all these cases above there is some conflict during the parallel execution of some actions. The avoidance of these kind of conflicts can be done by developing the SPN planning models as shown in Figures 5 and 6. Figure 5 shows a generic SPN plan model for two robots hands by synchronizing any possible conflict of actions during the parallel and concurrent execution of the plans. Note that the thick transitions Ti, represent stochastic time delays due to conflicts, while the thin ones represent direct transition with zero delay for the execution of a particular action.

SPN's Main Places and Transitions (actions)

P0 : starting place of SPN;

P1 : one or more objects are available;

P2 : assignment of the locations status;

P3 : robot hand H2;

P4 : 3-D workspace for execution of the plans;

P5 : robot hand H1;

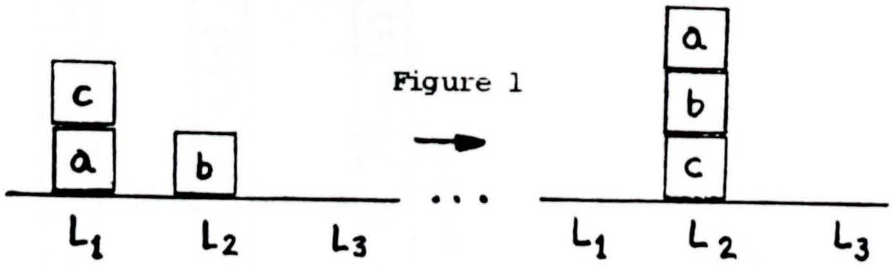


Figure 1

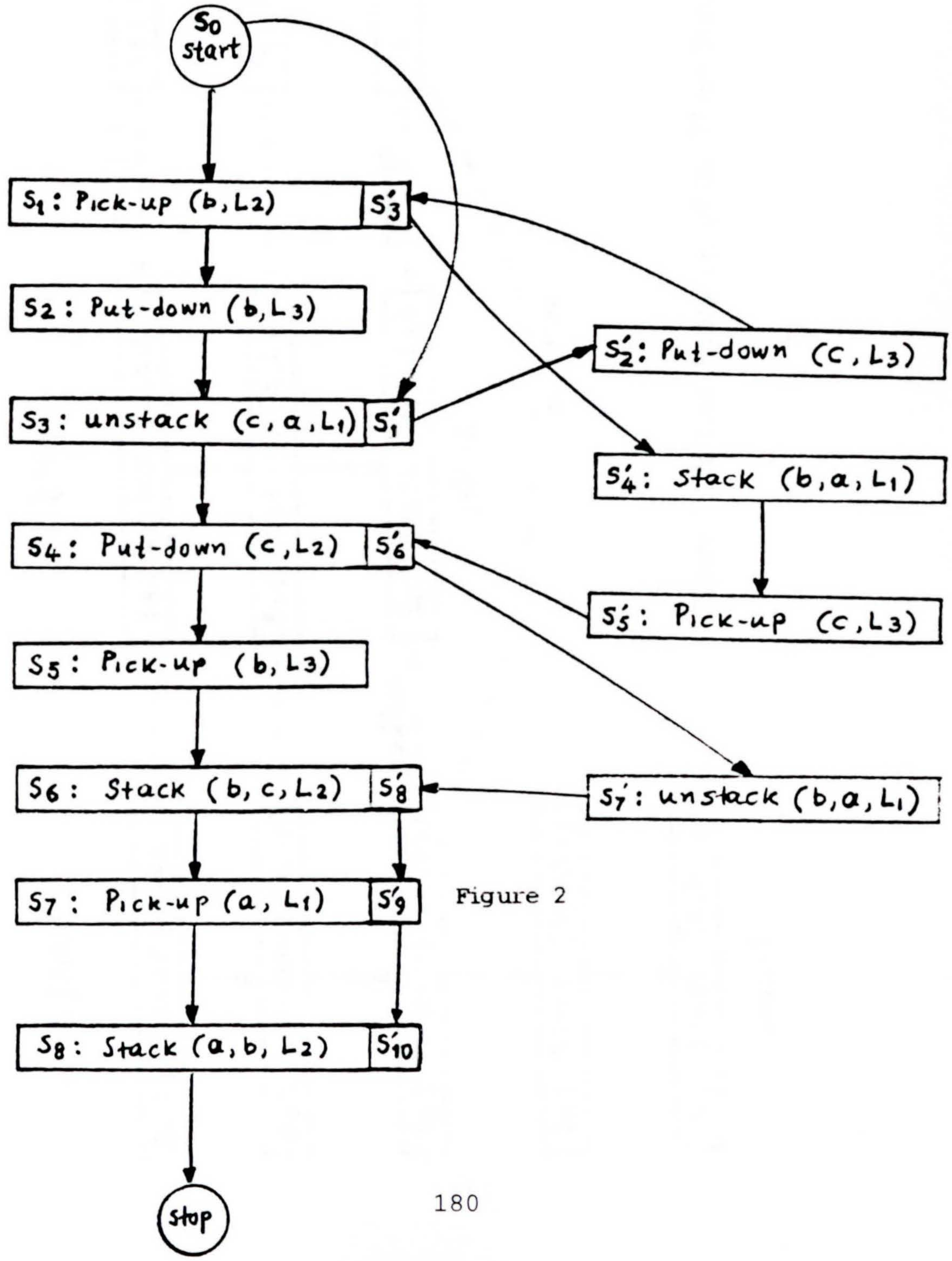


Figure 2

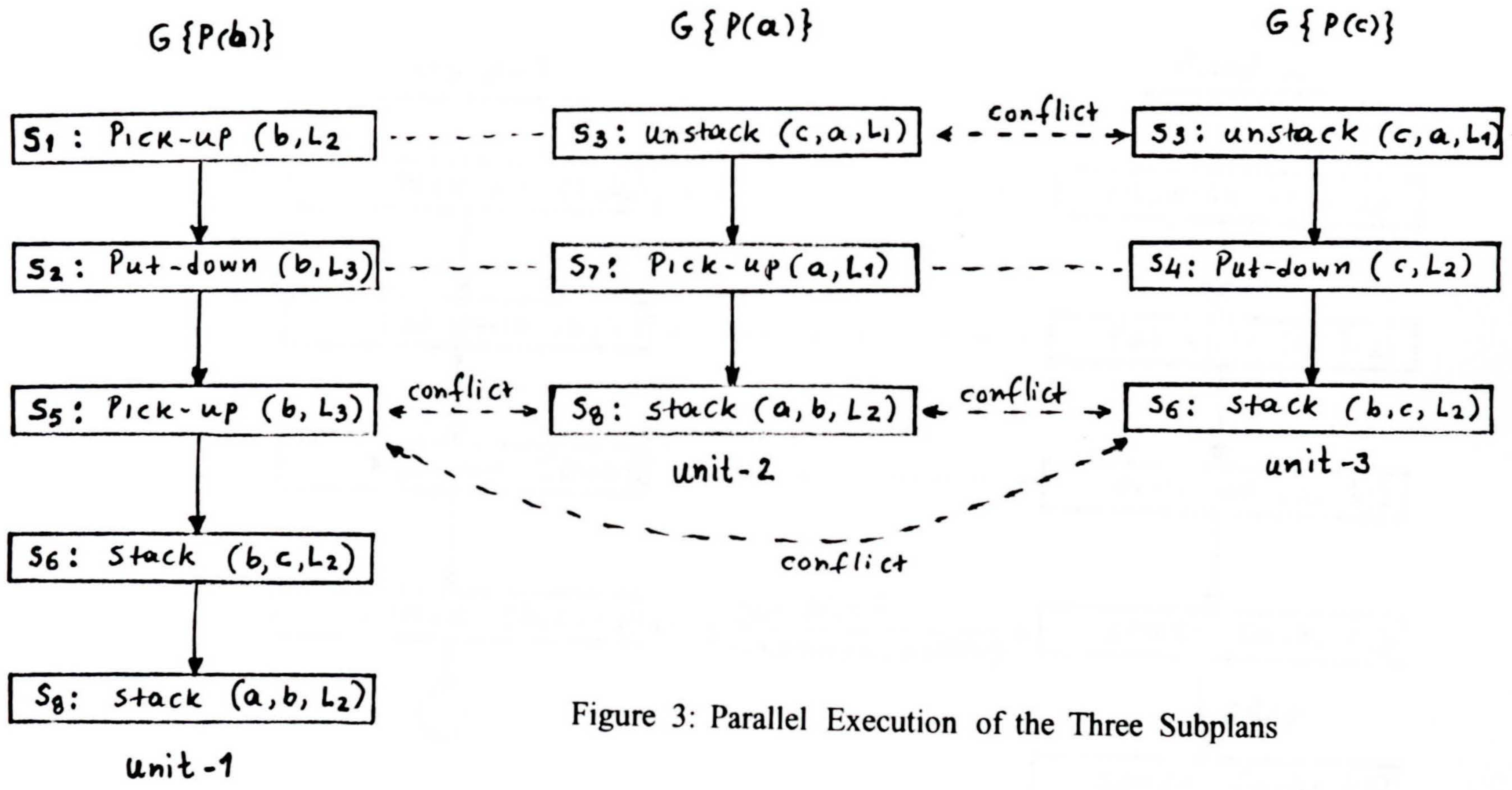


Figure 3: Parallel Execution of the Three Subplans

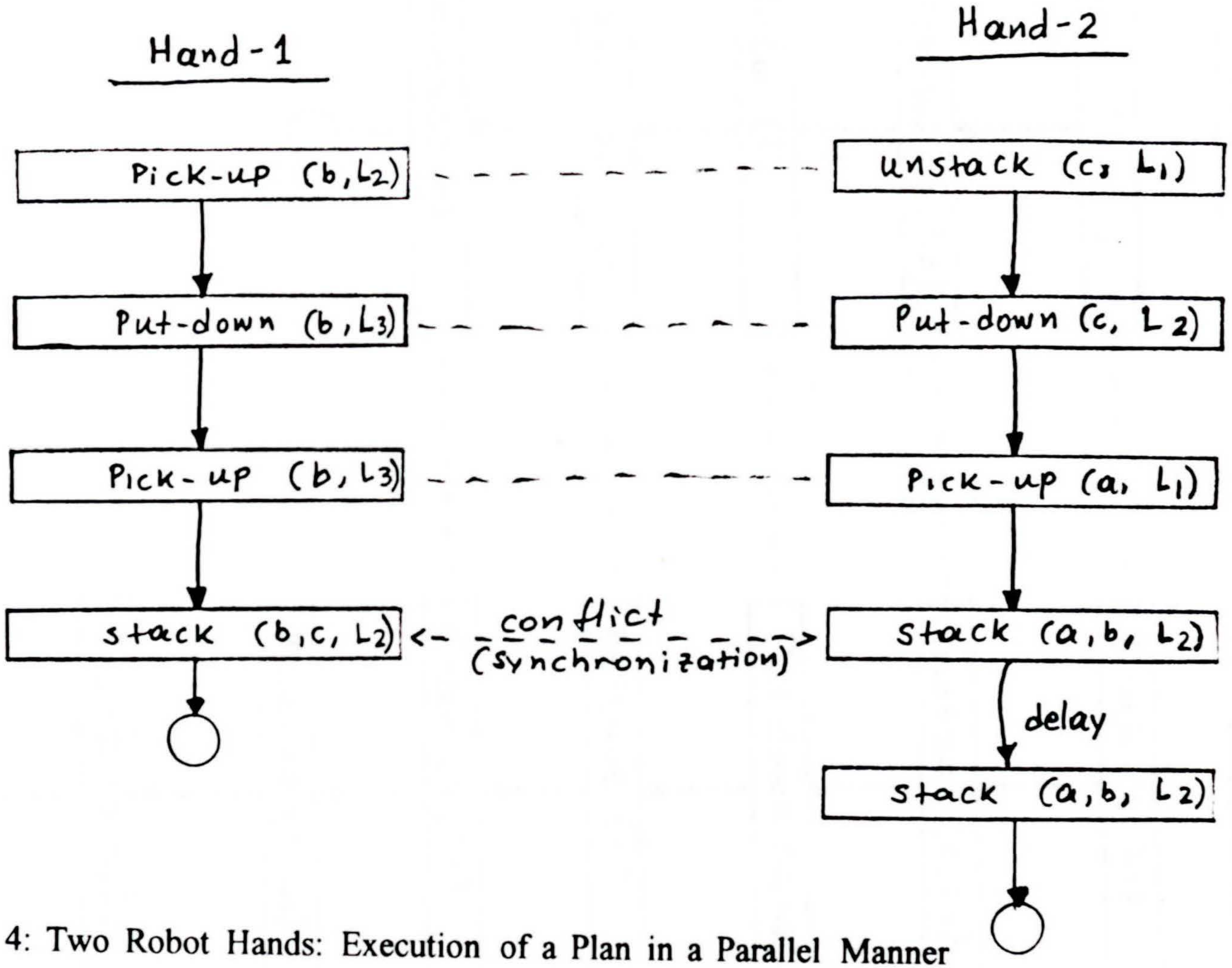


Figure 4: Two Robot Hands: Execution of a Plan in a Parallel Manner

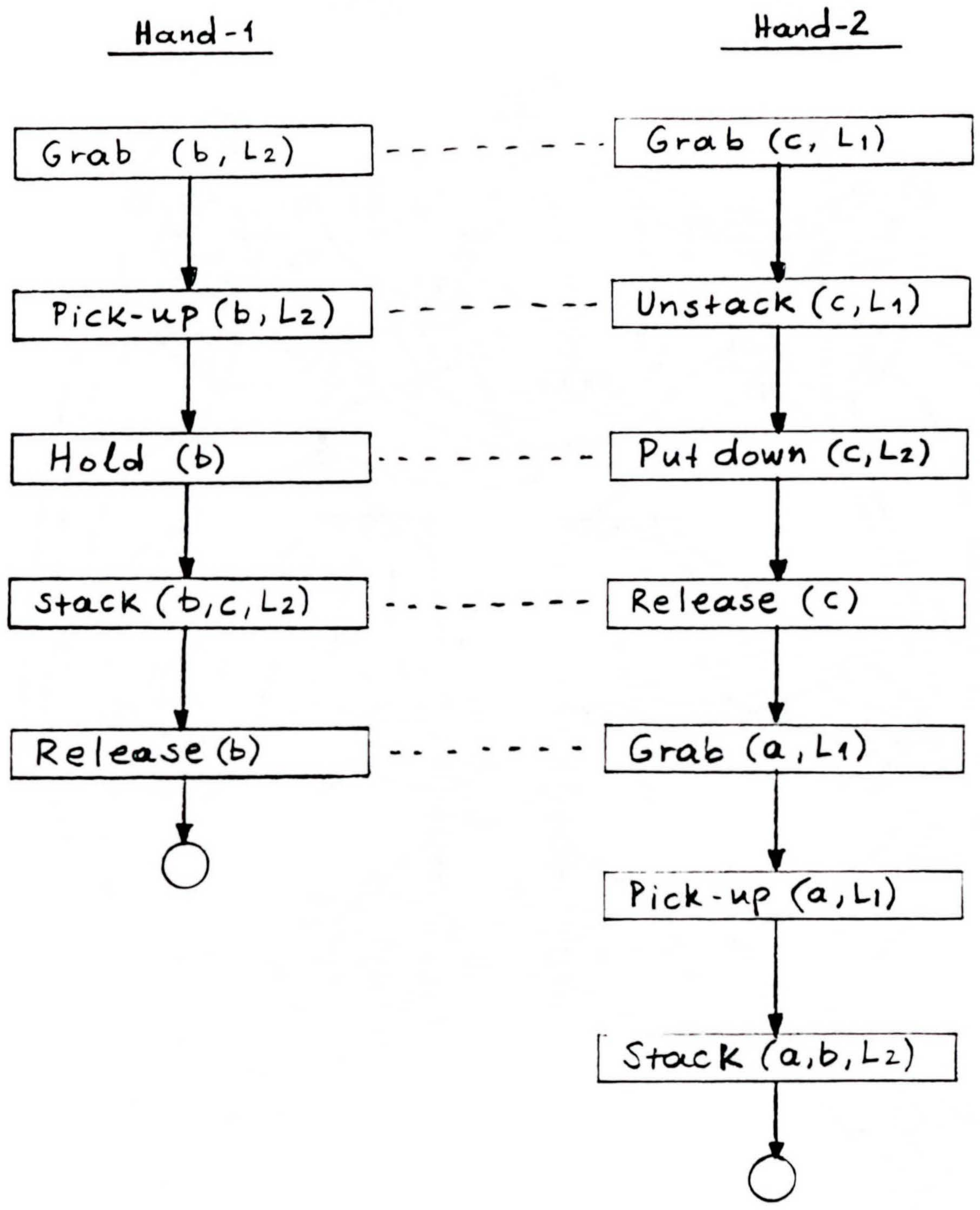


Figure 5: Two Robot Hands Execute a Parallel Plan Without Conflict

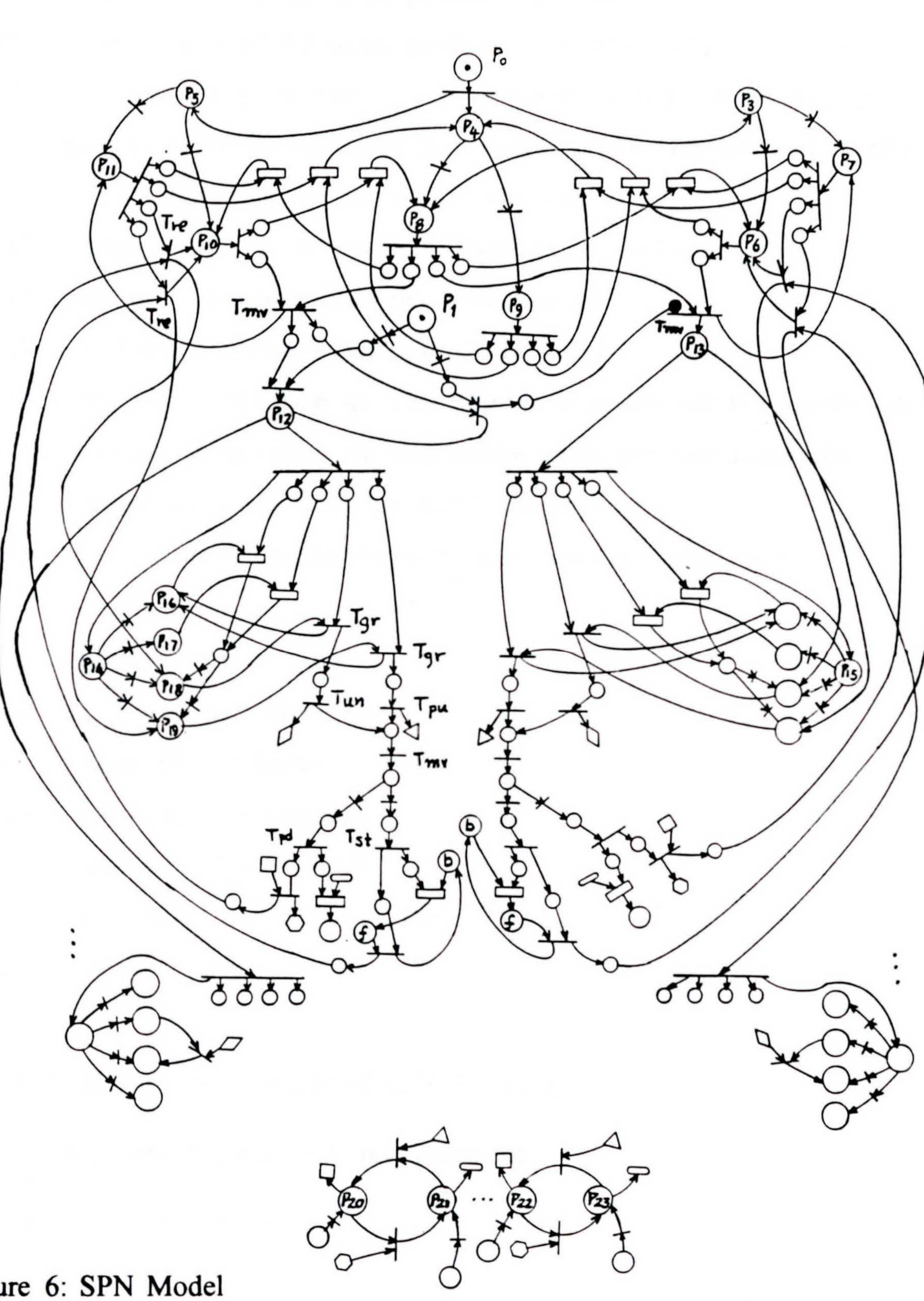


Figure 6: SPN Model

P6 : free state of H2 with probability p1;

P7 : busy state of H2 with probability q1=1-p1;

P8 : availability of the 3-D workspace with probability p2;

P9 : unavailability of the 3-D workspace with probab. q2=1-p2;

P10 : free state of H1 with probability p3;

P11 : busy state of H1 with probability q3=1-p3;

P16 : object grabbed with probability p4;

P17 : object blocked with probability q4;

P18 : object available at the top of a stack with probab. r4;

P19 : object available at the table with probability u4;

P20 : location free with probability p5;

P21 : location unavailable with probability q5=1-p5;

Tmv : move a robot hand;

Tre : release the object;

Tgr : grab the object;

Tpu : pick-up the object;

Tun : unstack the object;

Tpd : put down the object; and

Tst : stack the object;

6.4 An Illustrative Example of SPN Planning with a Self Organizing Neural Network

6.4.1 The Example Scenario

In this section a neural network model is used for the efficient search of the SPN framework by using the simple example of the previous section. In particular, an ART1 self-organizing network for categorizing binary vectors[11,12] is employed to extract an efficient strategy for a block manipulation problem by using two robotic arms (hands), three blocks and three block positions. The ART1 network is trained to always achieve the same goal, a above b above c on L2, within a minimum number of time steps and from any starting configuration.

As a strategy supervisor, the neural network decomposes a larger strategy that may not be intuitive into a hierarchy of smaller goals. The most obvious example of the benefit of strategy decomposition is when Hand 1 grasps its first block and must decide to either wait or proceed toward one of the two other positions.

Sixteen states and nine previous actions were arranged in a vector and coded as "1" for valid and "0" for not valid for a particular time step. For a given end goal, the network was trained with four starting states:

(c above a above L1, b above L2), (a above b above L2, c above L3), (a above c above b above L2), and (b above a above c above L3).

The goal state for Test Scenario 1 (Figure 7) was for the network to achieve (a above b above c above L3), and the goal for Test Scenario 2 (Figure 8) was (a above c above L1, b above L2), to be completed directly after Scenario 1.

```
The states and actions vectors are:

Test Scenario 1 and 2 States:
S(1)  Hand1 available
```

S(2) Hand2 available

S(3) Hand1 waiting

S(4) Hand2 waiting

S(5) L1 Position clear above

S(6) L1 Position available

S(7) L2 Position clear above

S(8) L2 Position available

S(9) L3 Position clear above

S(10) L3 Position available

S(11) Block a clear and available

S(12) Block b clear and available

S(13) Block c clear and available

S(14) Hand grasping or releasing Block a

S(15) Hand grasping or releasing Block b

S(16) Hand grasping or releasing Block c

Test Scenario 1 Possible Actions:

A(1) Move c to Not L2

A(2) Move Block (a or b) to Not above c

A(3) Move a to Not L2

A(4) Move b to Not L2

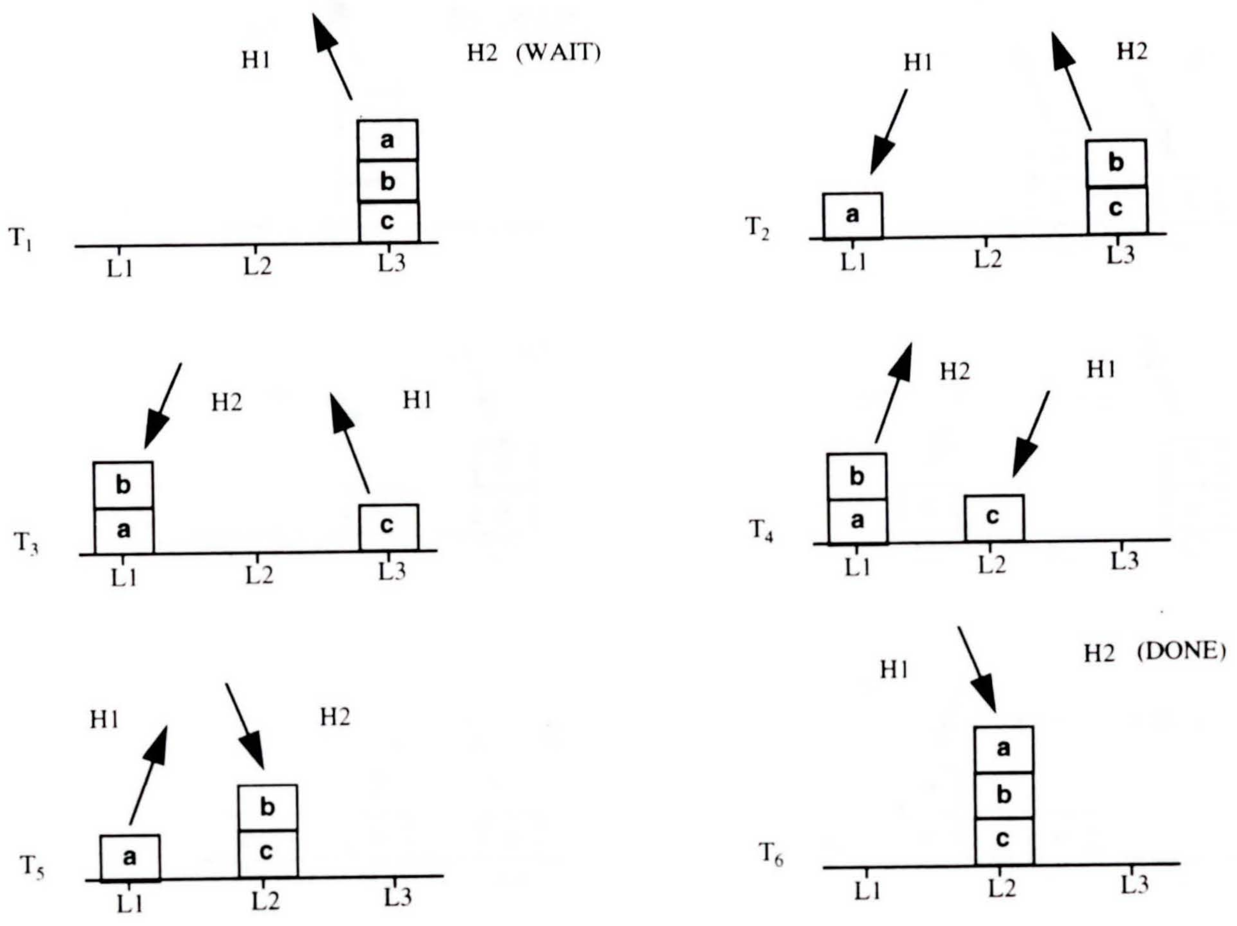
A(5) Move c to L2

A(6) Move a to Not above b

A(7) Move b to above c

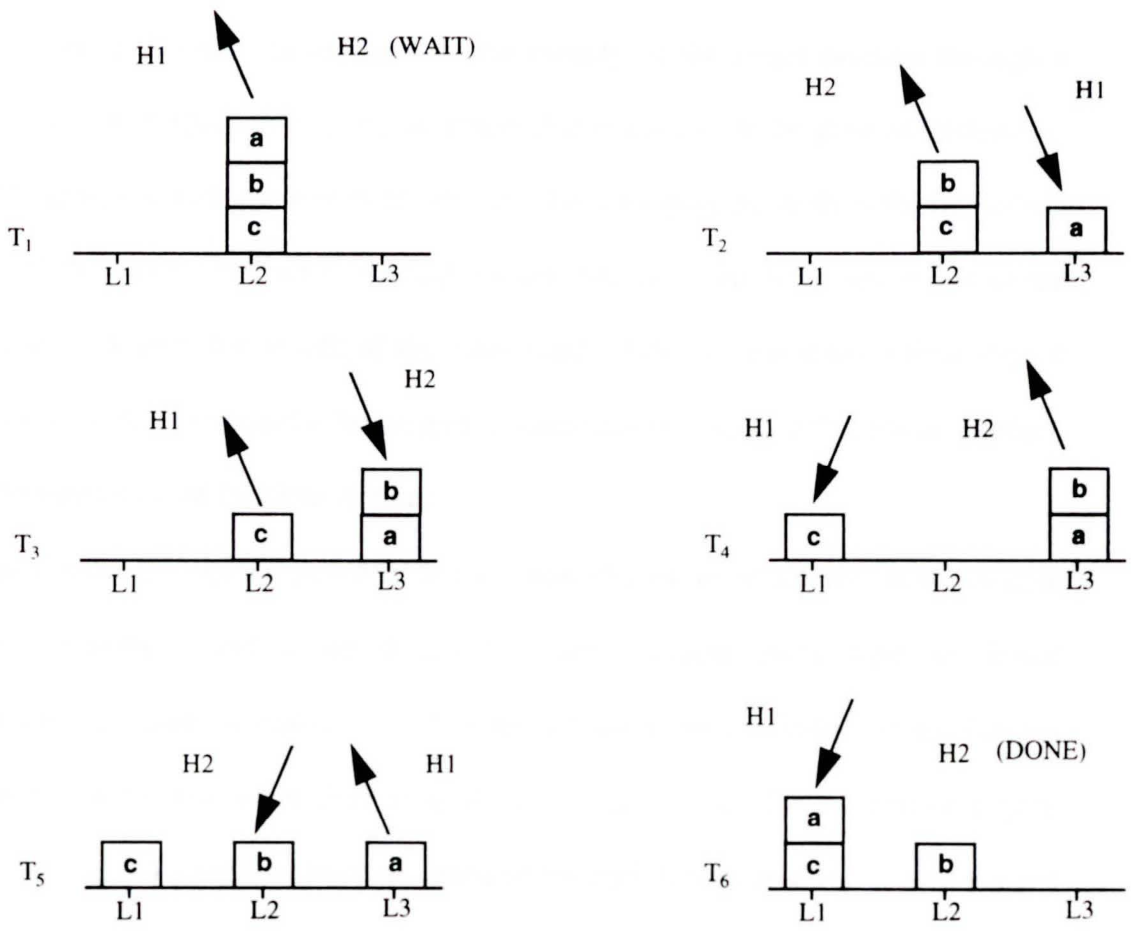
A(8) Move a to above b

A(9) Wait



Test Scenario 1

Figure 7



Test Scenario 2

Figure 8

A block is considered unavailable if grasped by a hand or covered by another block. A grasp requires movement into the vicinity of the target position through a possibly busy workspace or to grasp an object that is not free to be grasped. Releasing, similar to grasping, includes movement through the workspace away from the immediate vicinity of the block. The act of waiting implies that the active hand moves out of the immediate workspace for benefit of the other hand. After a hand waits a time step, it then reviews its recommended substrategies (actions) and chooses one that has no conflict.

6.4.2 Determination of Possible Actions

A minimum number of possible actions was chosen to attain the next possible subgoals. Actions 1 and 2 are designed to move blocks away from an initial configuration. A common strategy in a stacking "situation" for a next-to-bottom object is to place it in a location other than its goal state to clear space for the bottom object. Actions 1, 3, and 4 have this logic designed in for each block, as there is a very good chance that a block must end up on top of another in the desired final configuration. Actions 5 through 8 provide the robot arms an opportunity to stack blocks in relation to other blocks or locations as referenced to their final desired configuration. Action 9 is the action of waiting, mandatory for multiple hand scenarios. Note that the actions are not specific as to location other than to their final destination. This provision should be able to accommodate more or fewer spare locations in a future scenario.

6.4.3 Logic Supervisor Choice of Actions

A vector representing the current state of the scenario is introduced to the network, which produces a recommended action category. After a next action has been chosen by

5	4,5	5
6	6,7	8
7	2,8,9	9

It is interesting to note that Hand 2, which always follows Hand 1 at each time step, acquired a category 7 devoted to waiting.

Test Scenario 1 Resulting Actions:

Robot Hand and Step	Resulting Category	Recommended Actions	Chosen Actions
H1-1	4	2,7	2
H2-1	3	2,3,5,7,8	2 -> 9
H1-2	-	(2,7)	2
H2-2	-	(2,3,5,7,8)	2
H1-3	5	4,5	5
H2-3	-	(2,3,5,7,8)	2
H1-4	-	(4,5)	5
H2-4	3	3,5,7,8,2	7
H1-5	7	8,9,2	8
H2-5	-	(3,5,7,8,2)	7
H1-6	-	(8,9,2)	8

Robotic Hand and Step	Logic Followed
H1-1	a, above c, is available
H2-1	L3 is busy with Hand 1
H1-2	a placed on L1 (Not L2)
H2-2	b is now grasped
H1-3	b is busy, c grasped
H2-3	b is placed on L1
H1-4	c is placed on L2
H2-4	a not clear, c busy, b clear
H1-5	a clear
H2-5	b is placed on c
H1-6	a is placed on b

Test Scenario 2

The previous example is extended now to move to a new end state. The available actions are modified to accommodate the new goal states and their desired locations. The new goal state is to be achieved by moving from the previous goal state (a above b above c above L3) to the new goal (a above c above L1, b above L2).

Test Scenario 2 Possible Actions

A(1) Move c to Not L1

A(2) Move (a or b) to Not above c

A(3) Move a to Not L1

A(4) Move b to Not L2

A(5) Move c to L1

A(6) Move a to Not above c

A(7) Move b to above L2

A(8) Move a to above c

A(9) Wait

Test Scenario 2 Resulting Actions

Robotic Hand and Step	Chosen Actions	Human Actions	Logic Followed
H1-1	3	3	a is directed away from destination
H2-1	2->9	2->9	Hand 2 must wait for Hand 1
H1-2	3	3	a is placed on L3
H2-2	2	2	b begins toward default L1
H1-3	5	5	c is directed toward its destination
H2-3	2	2->9	b is redirected by Hand 1 with c
H1-4	5	5	c is placed on L1
H2-4	7	7	b is directed toward its destination
H1-5	8	8	a is directed toward its destination
H2-5	7	Done	b is placed on L2
H1-6	8	8	a is placed on c

6.5 Conclusion

In this chapter a stochastic Petri-net planning model and a neural network were used in a synergetic way for the selection of the appropriate actions to achieve the goal state for the block manipulation problem. More specifically, the SPN plan model was generated for the development of the appropriate framework on which the neural network was used as an efficient search planning strategy. In the scenarios used as illustrative examples it was observed that, in Scenario 2, Hand 2 (at Test Step 2) had more than one destination option. Given no other options the default is set at (L1 before L2 before L3). However, if Hand 2 is in the process of placing a block on a space that Hand 1 intends as a target location for the block it is grasping, Hand 2 will go to another location which is still acceptable to the action constraints (time step T3).

This planning methodology (at the neural network search level) has a preoccupation with completing a subgoal task, which it has been given, interrupting with a wait step only when a location is being occupied by the other hand. An extension of this work is to change plans or paths midway through a task execution as the state of the process changes, although in the example above if a human's attempts to save steps does not decrease the overall time to complete the task. Another future extension of this work is the use of feedforward diagnostic nodes, which may be added to aid the next step strategy of a fuzzy supervisor.

References

[1]E.D.Sacerdoti, A structure for plans and behavior, **American Elsevier**, 1977.

[2]A.Tate, Generating Project Networks, **Proc. IJCAI**,1977, pp.888-893.

[3]S.A.Vere, Planning in time: windows and patterns for activities and goals,**IEEE T-PAMI**,1983, pp.246-267.

[4]D.E.Wilkins, Domain independent planning: representation and plan generation, **Artificial Intelligence**,22, 1984, pp. 269-301.

[5]D.Chapman, Planning for conjunctive goals, **Artificial Intelligence**,32, 1987, pp. 333-377.

[6]D.Corkill, Hierarchical planning in distributed environments, **Artificial Intelligence**, 1979, 168-175.

[7]E.D.Sacerdoti, Planning in hierarchy of abstraction space, **Artificial Intelligence**,5,1974, pp.115-135.

[8]S.Lee and K.Chung, Resource oriented parallel planning, **IJAIT**,1,1,1992, pp.85-115.

[9]A.Passino and P.Antsaklis, Planning via Heuristic Search in a Petri-net framework, **Proc.IEEE Conf. on Intelligent Control**, 1988, pp.350-356.

[10]M.Genesereth and N.Nilson, **Logical Foundations of AI**, M.Kauffmann Pub.,1987.

[11]S.Grossberg, **Studies of the Mind and Brain**, Drodrecht, Holland: Reidel Press, 1982.

[12]T.Kohonen, **Self-Organization and Associative Memory**, 2nd Edition, Berlin: Springer-Verlag, 1987.

[13]N.Bourbakis, A neural knowledge base for high performance processing and planning, **IJAIT**,3, 4,1995.

[14]C.A.Petri, **Communication with Automata**, PhD thesis, RADC-TR-65377,NY Jan.1966.

[15]L.J.Peterson, **Petri-net Theory and Modeling of Systems**, Englewood Cliffs, NJ: Prentice Hall 1981.

[16]G.Sarridis and F.Wang, A model for coordination of intelligent control using Petri-nets, **IEEE Symposium on Intelligent Control**, VA 1988, pp.28-33.

[17]F.Wang,K.Kyriakopoulos, T.Tsolkas and G.Sarridis, A Petri-net coordination model of intelligent mobile robots, **IEEE Transactions on Systems, Man, and Cybernetics**, 21,4,1991, pp. 777-789.

[18]Bourbakis, N., and A. Tascillo, "An SPN Planning Method Associated with Neural Nets," **Proceedings of the International IEEE Congress on Neural Networks**, Orlando, Florida, June 1994, pp. 2803-2807.

[19]Bourbakis, N. and A. Tascillo, "A Neuro-SPN Planning Approach for Multiple Robotic Arms," **IEEE Workshop on APS-95**, Pittsburgh, Pennsylvania, June 1995. Also submitted to **IEEE Transactions on Systems, Man, and Cybernetics.**

CHAPTER 7:

ROBOT DRIVER CONTROL WITH NEURAL NETWORKS

7.1 An Application of the Hierarchy Concept

This chapter is an application of the wheelchair robot hierarchy concepts to that of a vehicle driving on dynamometer rollers. Although modified for the practical considerations of the application, there are strong similarities between the hierarchies, showing that the approach is extendable to other areas. The author's contribution[14] is the addition of a fuzzy spectral filter, interpreted by radial basis function neural networks, as a diagnostic contributor to the original planning and control hierarchy. The method is not only able to categorize vehicles with signatures that do not look similar to the human observer (especially due to noise), but it can also properly identify parameters of a vehicle it has never seen before. The fuzzy spectral filter concept can be extended to applications such as gamma ray categorization and robot navigation in harsh or unexplored environments.[15]

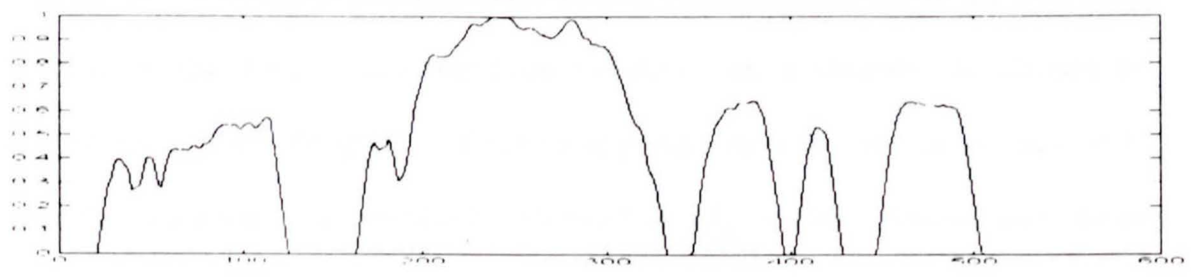
7.2 Problem Scenario

Repeatability in vehicle testing on dynamometer rollers is critical for providing automotive engineers with a consistent correlation to real world conditions. Commercially available robot driver controllers have been successful at following a velocity trace using PID and H_∞ based approaches, but the throttle, clutch, and braking behavior to achieve the desired velocities has not closely resembled that of a human driving the same trace. In addition, the robots have difficulty driving automobiles significantly different in performance characteristics than those they were developed for. In response to these challenges, neural networks are used in the diagnostic adaptive control of a robot driver which must minimize dynamometer slip, insufficient braking, and other errors due to automobile, dynamometer, and environmental variation. The resulting robot driver and controller can be employed to more accurately compare test sites, automobiles, and engine controller strategies. By reducing the variability from automobile to automobile and test run to test run, the resulting controller models can be used in error propagation studies to further reduce other sources of variability in the test process.

The goal of this effort is to minimize the error between a scheduled velocity trace (i.e., Figure 1) and the actual roll speed of a vehicle chassis dynamometer (dyno). Normally a human driver applies throttle, clutch, and brake movements in reaction to velocity error as viewed on a computer screen called the driver's aid. With time, a driver learns the inadequacy of reacting only to current speed error -- by recalling previous experiences with upcoming portions of the trace, the driver applies pedal movements in anticipation of a lag in the vehicle response. These lags can vary due to vehicle inertia (J), vehicle horsepower (HP), vehicle/engine temperature relative to a norm, and time the vehicle has been continuously testing on a given dyno. One example of an inertia, temperature, and time dependent error that requires on line driver adaptation is brake fade (Figures 2 and 3), shown here for Vehicle 1, a heavy rear wheel drive vehicle. If the driver finds that, due to heating of the vehicle and dyno, a greater brake force is necessary to slow a vehicle as a test progresses, he or she learns to apply the brake sooner and/or use less throttle as the next planned brake approaches.

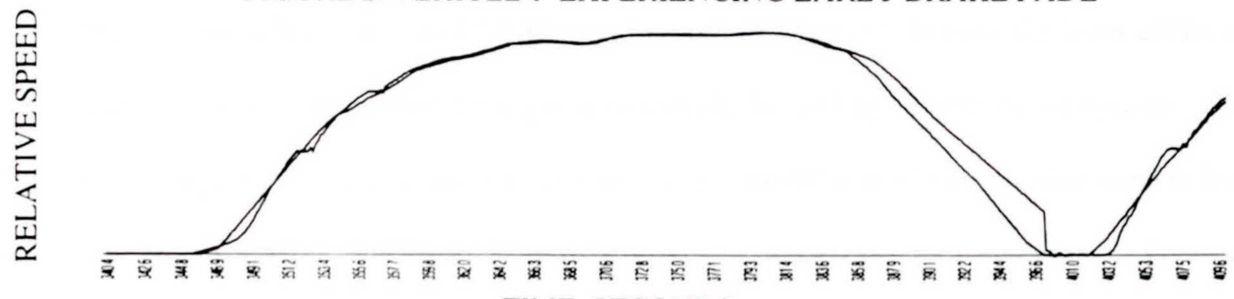
Two constraints are added to the problem of diagnostics and adaptive control in an environment where vehicles (and functional variations of vehicles) change several times daily on a given dyno. The first is to implement diagnostics and control that is easily integrated with, or using, hardware and software already within the laboratory. Extra equipment must be purchased, stored, and maintained. In addition, data acquisition at high sampling rates is not always available or desired in an industrial testing environment. Most data collection is not necessary at rates greater than one to ten samples per second, and greater sample rates have both processing and storage ramifications. Lower sampling rates, however, cannot accurately capture the behavior of systems which require better control of their transient responses. A method is desired that can reliably extract useful information from available data, as well as remain flexible to upgrades in rates of data acquisition and control. The second constraint is to avoid increasing vehicle preparation or test time. The desired method should be accomplished while a vehicle parameter is not under active control (such as at idle for throttle, or before deceleration for brake), without interaction with the vehicle's computer control (which has to be manually interfaced), and without installation of extra sensors during test preparation.

FIGURE 1: SCHEDULED SPEED FOR A GIVEN TEST



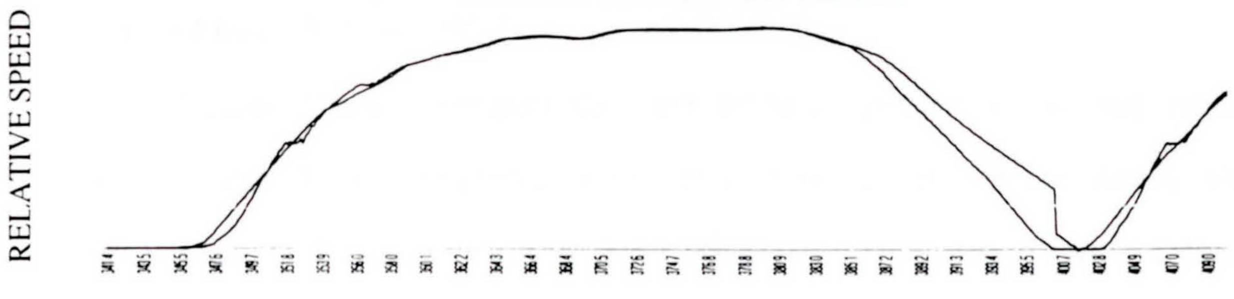
TIME, SECONDS

FIGURE 2: VEHICLE 1 EXPERIENCING EARLY BRAKE FADE



TIME, SECONDS

FIGURE 3: VEHICLE 1, ONE TEST BEFORE TEST WAS ABORTED DUE
TO FAILURE TO CATCH NEXT ACCEL



TIME, SECONDS

VEH. 2 DRIVER AVERAGE ERROR

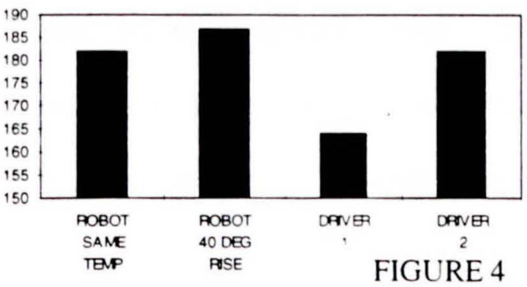


FIGURE 4

VEH. 2 ERROR STANDARD DEVIATION

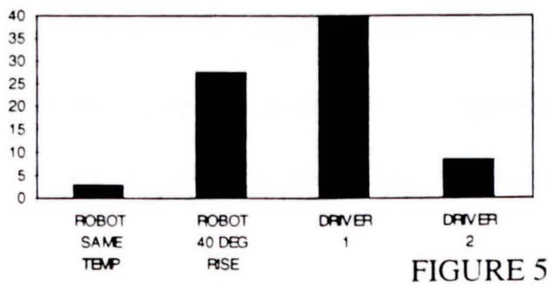


FIGURE 5

7.3 Current Controller Performance

Due to nonlinear time dependent phenomena, memorization of the appropriate gains for a PID or other fixed model-based controller are inadequate. In addition, any control model that is designed for this type of process must be stable across all possible operating conditions, but must also be able to adequately identify and transition to a more accurate set of controller gains for the new driver/dyno/vehicle state. Commercially available manual transmission driving robots have demonstrated acceptable performance for most vehicles under most laboratory driving conditions.[1,2] During the learn cycle, a control model is developed for a given robot/vehicle (and optionally dyno) system. To ensure repeatability under standard conditions, this model is used to drive that vehicle for all velocity traces irrespective of changing system conditions. This approach to control results in low standard deviation for the vehicle under the conditions the robot was trained, but repeatability degrades when tests are included where the temperature is increased from 75° F to 105° F.

Figures 4 and 5 compare the trace following performance of one robot versus Driver 1, an exceptional driver, and Driver 2, an average driver, all for Vehicle 2, a high performance front wheel drive. Figure 4 shows Driver 1 with the least absolute error summed second by second, followed by the Robot and Driver 2. Robot average error exceeded both human drivers once test room temperature was varied. The robot's "average" trace following error for these tests is attributable to a drive style programmed to optimize a wider range of parameters, only one of which is trace following. Although

Driver 1 was the more accurate driver, Figure 5 shows that his standard deviation was higher than the robot or Driver 2. This can be attributed to the fact that tests were run with Driver 1 before and after he went on vacation, while Driver 2 drove his tests within a single week. The robot demonstrated its advertised superior standard deviation at room temperature, but looked like it was coming due for a vacation when temperature was taken into account.

7.4 A Hybrid Controller Approach

The proposed approach to this problem is to extract a vehicle's state from its frequency signature while idling on the dyno just before the desired velocity trace begins. This state information is then fed into fast response combinations of neural networks[3] and fuzzy logic[4] schedulers which can stably adjust the weights of default controllers to gain a timely and accurate response to any given velocity command trace (Figure 6). Neural networks and fuzzy logic have already shown promise in diagnostics[5,6,7] and control.[8,9,10] A major advantage to implementing nonlinear representations in neural network or fuzzy logic form is the speed of computation -- a series of adds, multiplies, and conditional statements. Computations that might have required several iterative passes through formulae connected by several conditions and look up tables can be condensed into one forward pass through dedicated modules. The challenge of this effort is to extend these ideas to accurate, reliable operation in the industrial setting given real world constraints.

FIGURE 7: VEHICLE 1 DYNO TORQUE

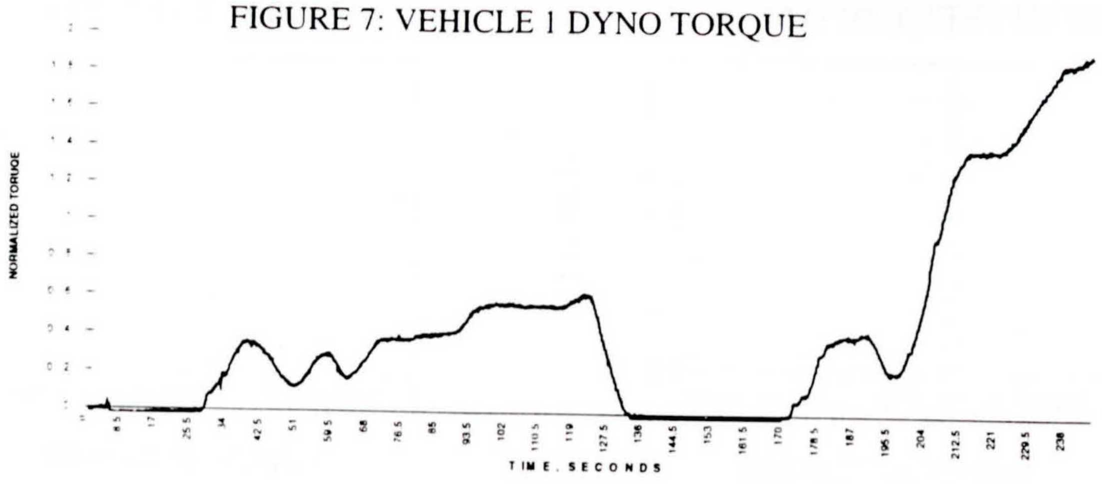


FIGURE 8: VEHICLE 2 DYNO TORQUE

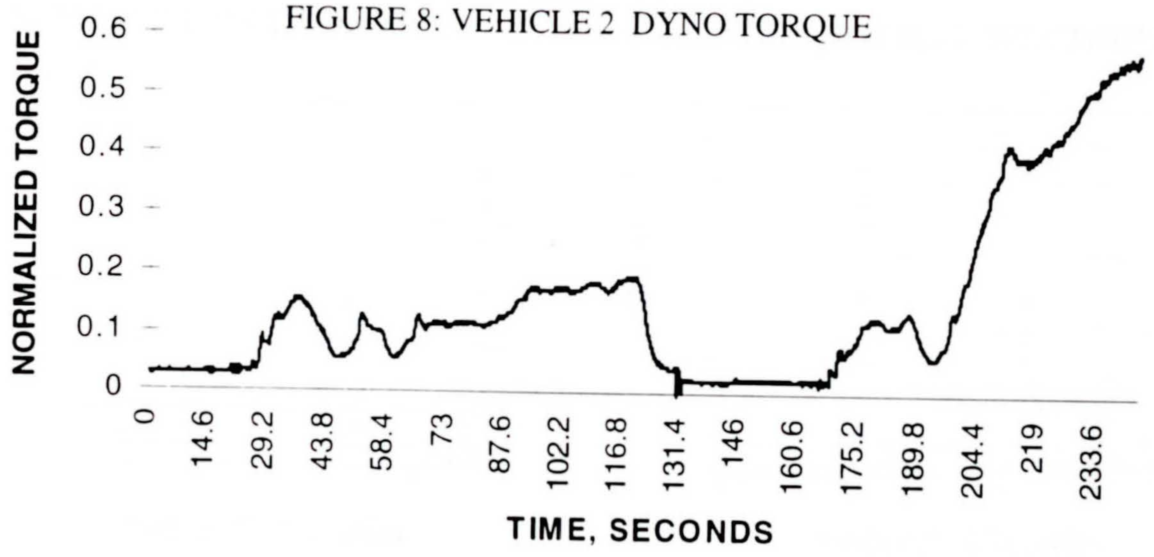


FIGURE 9: VEHICLE 3 DYNO TORQUE

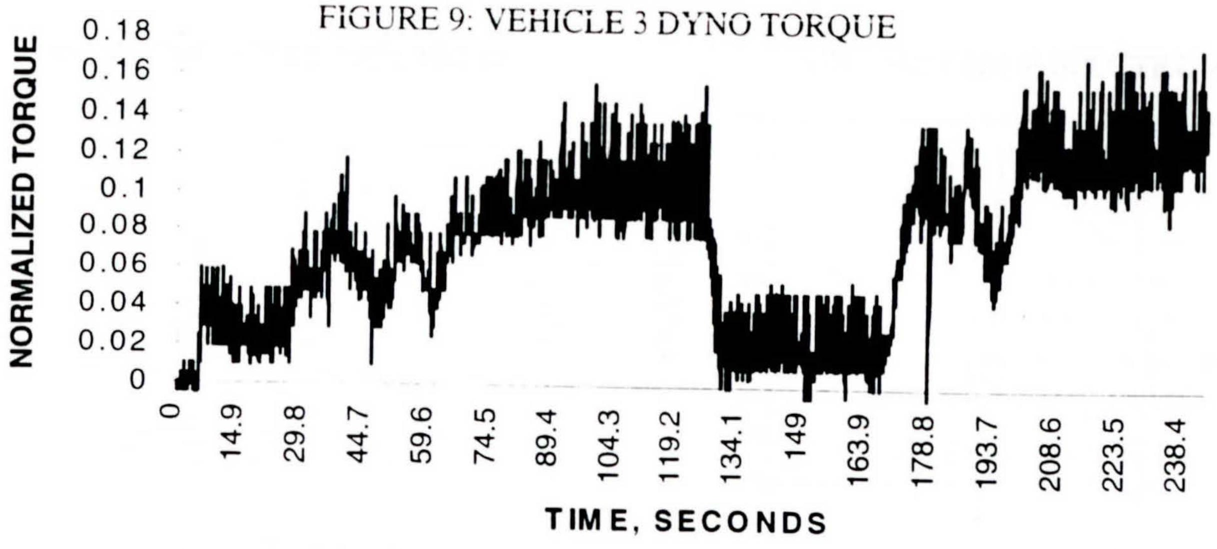


FIGURE 10

VEH 1 COLD IDLE SPECTRUM

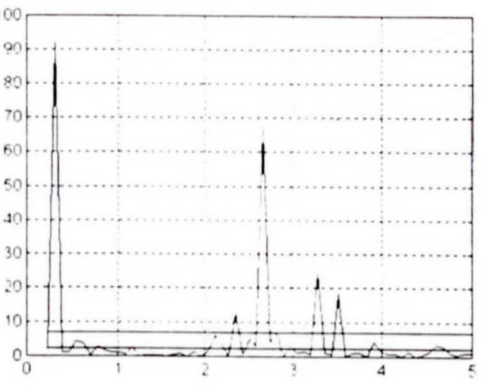


FIGURE 11

VEH 1 HOT IDLE SPECTRUM

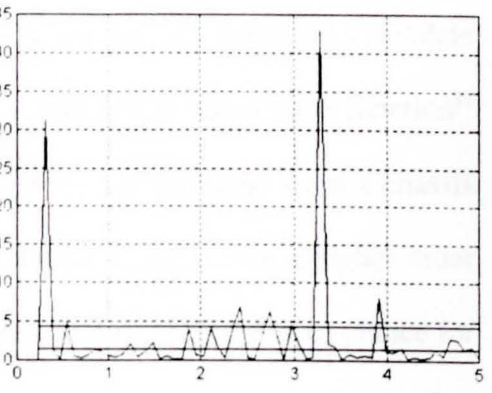


FIGURE 12

VEH 2 COLD IDLE SPECTRUM

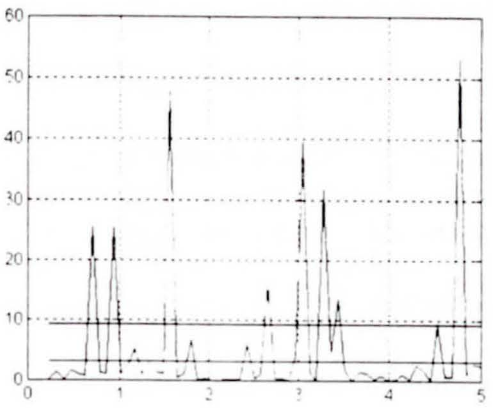


FIGURE 13

VEH 2 HOT IDLE SPECTRUM

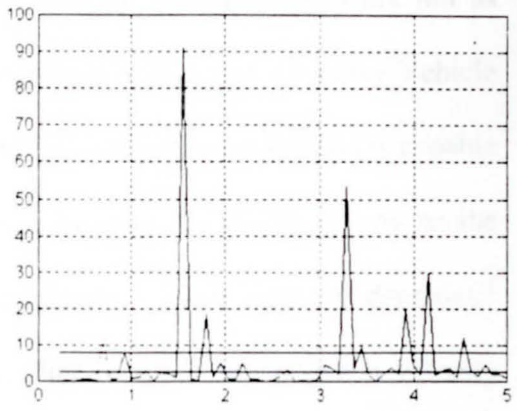


FIGURE 14

VEH 3 COLD IDLE SPECTRUM

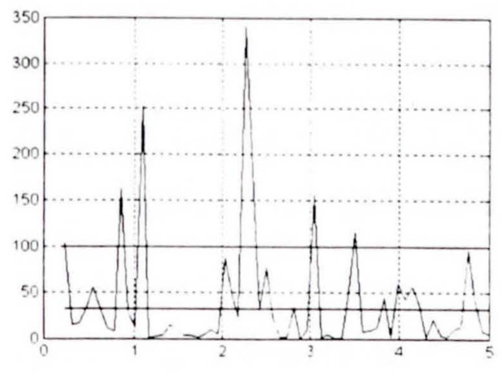
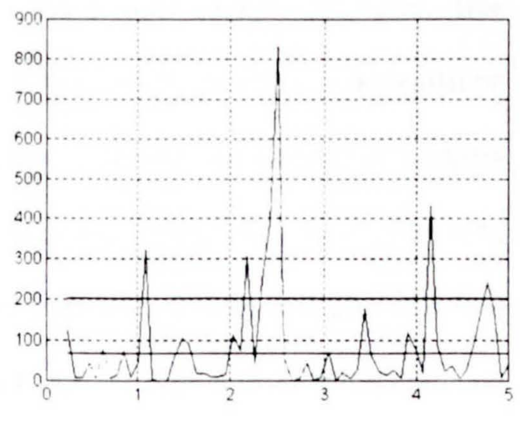


FIGURE 15

VEH 3 HOT IDLE SPECTRUM



7.5 The Fuzzy Spectral Filter (FSF)

The FSF, in the lower left corner of Figure 6, singles out weighted candidate frequencies for classification by the perceived vehicle state (PVS) radial basis function[11] neural network. Two signals, velocity and torque, are readily available from a chassis dyno, and are dependent upon the vehicle above it. Dyno torque, being a higher order signal, contains more useful information than roll speed, and was the signal of choice for this work. Figures 7 and 8 are typical examples of hydrokinetic dyno torque for the first 240 seconds of the Figure 1 desired velocity trace. Although it is obvious that a heavy vehicle will cause a larger dyno torque at nonzero velocity, the differences are not as obvious at idle. Additionally, in Figure 9 where amplitude analysis of the noisy Vehicle 3 (a light weight rear wheel drive) dyno torque might fail, the proposed method is capable of identifying the vehicle. The technique can also be extended to electric dynos, as the frequencies are similar, yet shifted by a fraction of a Hertz. Power spectral densities[12] are calculated for stretches of idle before ignition, after ignition, and between velocity "hills" along the desired trace as they occur. A power spectral magnitude response (PSMR) is found by dividing each idle spectrum by the first preignition spectrum in order to characterize the gain introduced by the running vehicle over background noise. Idle periods can vary significantly in length, from seconds to minutes. In order to demonstrate the proposed approach's ability to handle low spectral resolution, the number of samples for each idle spectrum was limited to less than 128 points. Preignition data, consisting of as few as 47 samples, was used in conjunction with idles before and after the first velocity "hill" to generate spectra such as those in Figures 10 through 15 (with common frequencies of large magnitude removed).

Due to the limited number of data points, the magnitudes and frequency locations of each peak can not be relied upon in a strict sense. High frequencies tend to be exaggerated, and low frequencies are lost in the noise background. In order to best extract significant frequencies for a given vehicle category, significant frequencies for representative vehicles are arranged in a reference vector, and gaussian membership functions are centered about each frequency. After the six largest magnitudes of a candidate spectrum are subtracted, a line that is three times the average value of the remaining magnitudes is used a minimum magnitude threshold for the selection of candidate frequencies. Frequencies which pass within the area of a membership gaussian are assigned a membership value and added to any previous membership values in that vector location. Nonzero membership values can then be summed across frequencies for a candidate vehicle (two for the example following), and compared to those for the other vehicles in the candidate vector, or in the complete controller implementation, left unsummed and entered into the PVS radial basis network for classification.

This example radial basis function was trained to determine a vehicle in question's relative temperature (cold (-10) or hot (10)), its normalized inertia (J/1000), and its dyno horsepower (HP), given a vector of frequency memberships for 5 representative vehicles (Vehicles 1, 2, and 3 plus a medium performance front wheel drive). This neural net was then tested on a fifth vehicle, a heavy high performance rear wheel drive.

Training Example Inputs and Outputs:

Inputs (Frequency memberships)	Outputs (RVT,J,HP)
[1.74 .35 .05 0 0 .91 .9 0]	[-10 8.0 19.4]
[.84 1.02 .05 0 0 .91 .9 0]	[10 8.0 19.4]
[.84 .04 .46 0 0 .75 0 0]	[-10 8.0 19.4]
[.84 .35 .44 0 0 .75 0 0]	[10 8.0 19.4]
[0 .02 .44 1.0 0 .76 0 0]	[-10 3.25 7.0]
[0 .02 1.0 1.0 0 .12 .11 0]	[10 3.25 7.0]
[0 .35 .37 .37 .44 0 0 .02]	[-10 3.25 7.0]
[0 0 1.37 .03 .97 .11 0 .35]	[10 3.25 7.0]
[0 .35 .07 0 .97 1.11 0 .35]	[-10 3.375 6.2]
[0 0 .07 0 .97 1.11 0 .35]	[10 3.375 6.2]
[0 .02 .79 0 .97 .76 .84 .35]	[-10 3.375 6.2]
[0 0 .79 .02 .97 .75 .84 .38]	[10 3.375 6.2]
[.15 .02 .02 0 0 .01 .75 0]	[-10 3.625 8.1]
[0 1.37 0 .02 0 0 0 0]	[10 3.625 8.1]
[0 .04 .44 0 .4 .75 .84 .97]	[-10 4.25 8.9]
[.15 .02 0 .73 .40 0 0 .97]	[10 4.25 8.9]
[0 .02 0 .35 .97 .2 .84 .38]	[-10 4.25 8.9]
[0 0 0 .35 0 .2 .84 .03]	[10 4.25 8.9]

Test Inputs	Network Outputs	Actual Values
[.99 0 0 0 1.32 0 1.7 1.32]	[-10 3.9818 8.9867]	[-10 4.25 8.9]
[.99 0 0 0 .35 0 1.7 .97]	[10 4.2844 9.7883]	[10 4.25 8.9]

Considering that the examples are sparse at the high weight end of the cars (with only one vehicle with higher values), the approximations for the new Vehicle 5 examples are closest to their actual values, despite the seemingly dissimilar membership vectors. The RVT outputs are automatically rounded to their relative extreme of -10 or 10.

7.6 Radial Basis Functions for Categorization

Radial basis functions form a nonlinear mapping of a parameter space with (generally overlapping) gaussian functions. Given an input vector to be classified, the outputs of these gaussians are weighted and summed to produce a suggested category value. The outputs of the PVT network are perceived inertia (J), dyno horsepower (HP), and relative vehicle temperature (RVT). RVT can be determined by the relative drop in magnitude of a particular frequency for each vehicle. An anticipated throttle lag (ATL) radial basis function network uses these outputs as inputs to produce a suggested lag, or time before change in scheduled speed to apply the throttle, as demonstrated below.

This example radial basis function was designed to be able to predict throttle lag from scaled inertia, scaled horsepower, and dyno type (-10 for electric, +10 for hydrokinetic) from only a few examples. All examples were for warm engines. Low lag is the time between first throttle motion and some nonzero dyno roller velocity, and high lag is the time between first throttle motion and a major change in dyno roller velocity. The gaussian layer weight matrix became a 4x3 (bias weights 4x1), and the linear layer weight matrix became a 2x4 matrix (bias weights 2x1). For such a small training set the results look promising.

Training Example Inputs and Outputs:

Inputs (Dyno type, Inertia, Horsepower)	Outputs (Low lag, High Lag)
[-10 3.375 6.2]	[.65 .85]
[-10 8.0 19.4]	[.415 1.92]
[10 8.0 19.4]	[.40 1.01]
[10 3.25 7.0]	[.72 .85]
[10 4.0 9.2]	[.60 1.4]

Test Inputs (For vehicles with learned dyno counterparts)	Network Outputs	Actual Lags
[10 3.375 6.2]	[.6005 .9031]	[.60 1.2]
[-10 3.25 7.0]	[.5587 .9084]	[.66 .85]

(For a vehicle never seen before)

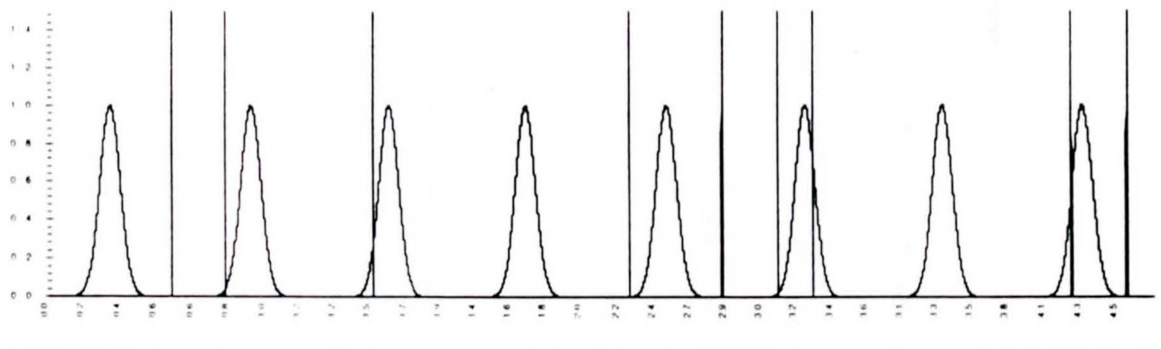
[-10 4.25 8.9]	[.4009 1.0094]	[.30 1.18]

7.7 A Neural Diagnostic Controller

A neural network diagnostic controller was developed in previous work[7] to diagnose a change in the plant and transition the default parent controller network to a more accurate child network. In this paper, practical implementations are considered, with sampling rate and timing restrictions. Instead of simply memorizing the children, the FSF, PVS, and ATL modules calculate just how much to modify the timing and amplitude of a parent network for a perceived drift in parameters. The child net configuration continues control for as long as the measured parameters are within operating tolerances. Learning from future experiences can be used to update each of these modules.

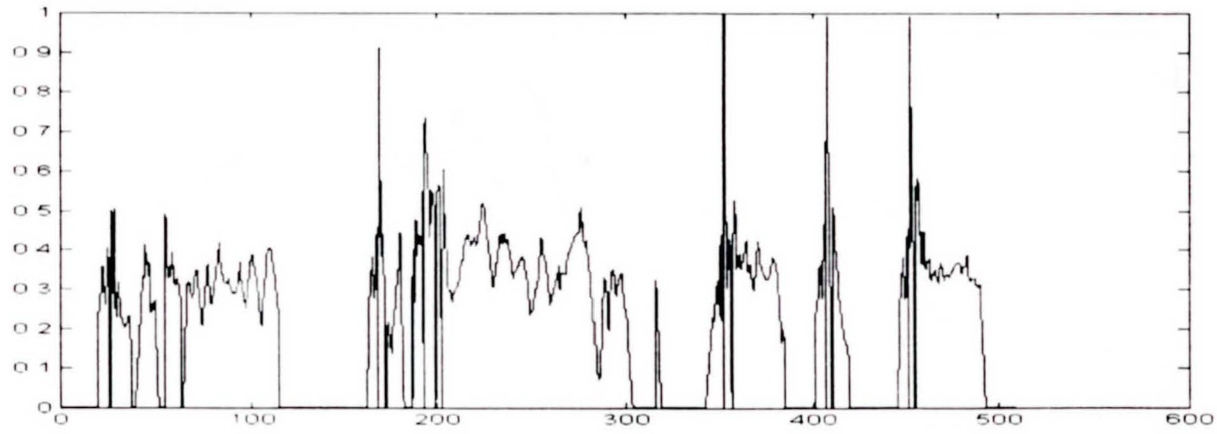
Figures 17 through 19 show typical throttle, brake, and clutch movements of a robot driver in a Mystique as it follows the scheduled velocity trace in Figure 1. Figures 20 and 21 show twenty seconds of FSFER net[13] suggested throttle and brake movements, before filtering. Given that a scheduled speed trace is known beforehand (and can be preprocessed during idle periods at the beginning of a test run and elsewhere), the rate of change of throttle determines the amount of smoothing for the suggested movement, and results in a throttle very close to the target. The extra fitting capacity in the network, although it exhibits higher order than at first necessary, allows it to accommodate

FIGURE 16: VEH 2. COLD IDLE CANDIDATE FREQUENCIES



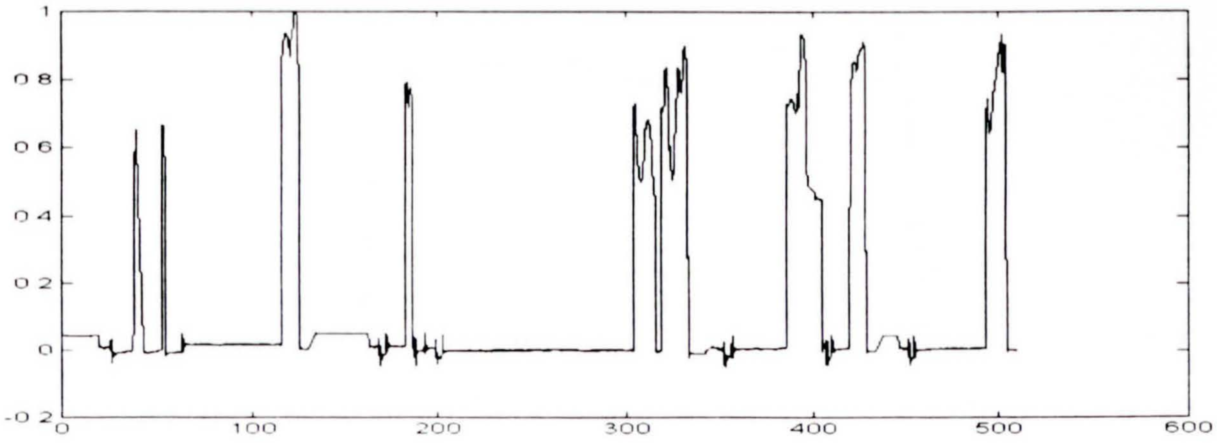
FREQUENCY. HZ

FIGURE 17: VEH 2, THROTTLE POSITIONS FOR THE TEST RUN



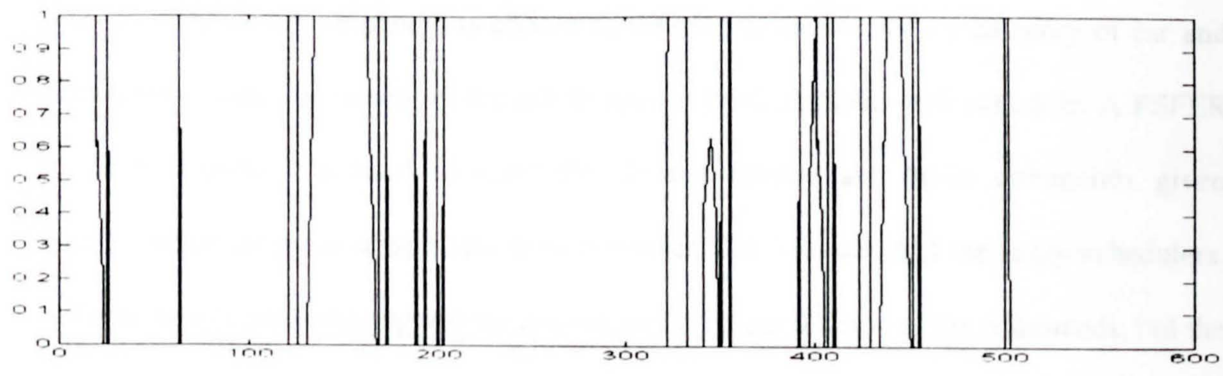
TIME. SECONDS

FIGURE 18: VEH 2, BRAKE POSITIONS FOR THE TEST RUN



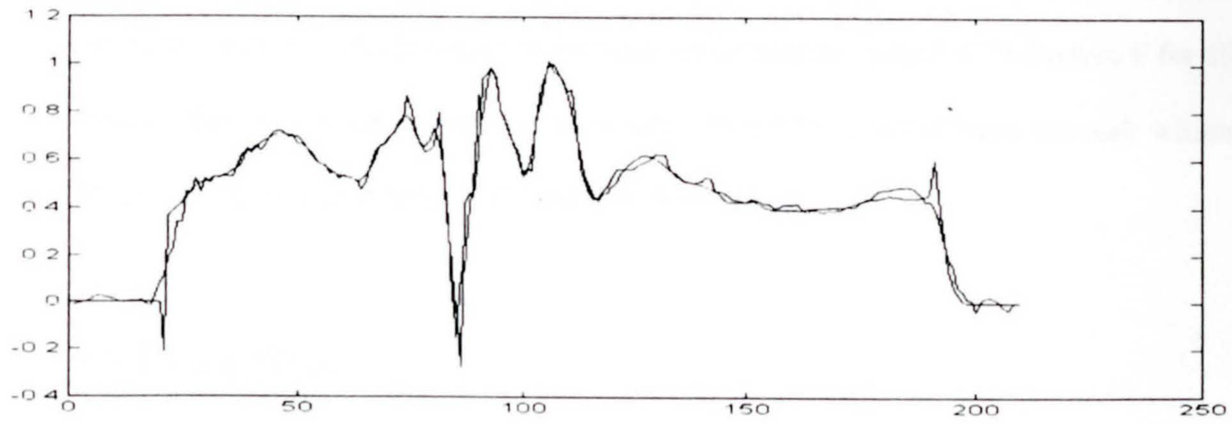
TIME. SECONDS

FIGURE 19: VEH 2. CLUTCH POSITIONS FOR THE TEST RUN



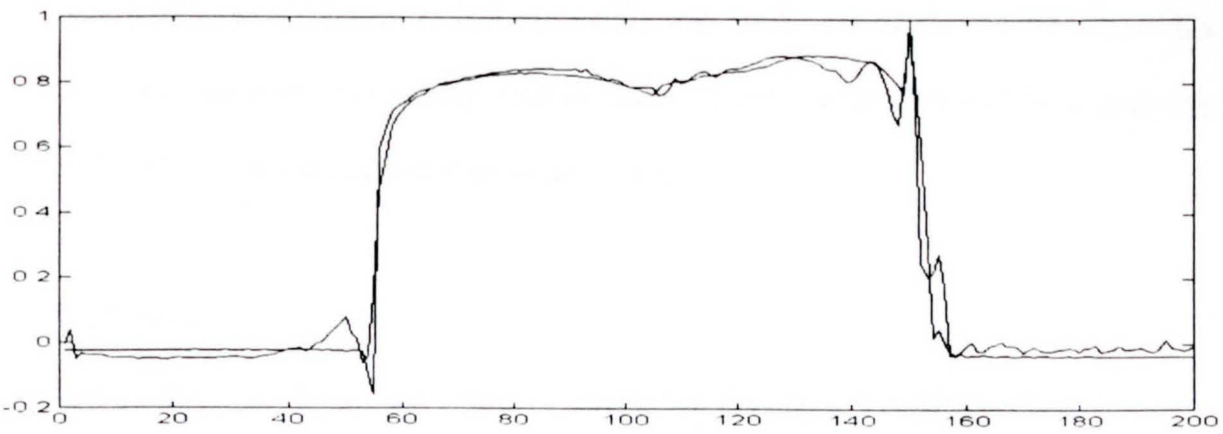
TIME, SECONDS

FIGURE 20: FSFER NET CONTROLLER SUGGESTED THROTTLE COMPARED TO ROBOT



SAMPLES AT 10 Hz

FIGURE 21: FSFER NET CONTROLLER SUGGESTED BRAKE COMPARED TO ROBOT



SAMPLES AT 10 Hz

anticipated changes in the system parameters.

The clutching pattern is chosen as the closest memorized by category of car and gear transition. Its timing is dependent upon a predetermined shift schedule. A FSFER neural network[13] is used to learn the desired throttle and brake commands given information by dyno sensors, the host computer, the ATL net, and the fuzzy schedulers. These neural networks suggest the magnitude and general shape of the commands, but the actual timing of a command relative to a change in desired speed is suggested by the most recent diagnosis of the ATL net.. Inputs to the radial basis net are the perceived inertia (vehicle weight), indicated horsepower (in reference to the dyno), and relative temperature of the vehicle/dyno combination versus having soaked at 75 degrees F for 10 hours. This perceived horsepower is in turn offered by a radial basis network whose inputs are fuzzy memberships of candidate frequencies.

7.8 Future Work

After more data is gathered, the new control strategy will be implemented as a separately compiled subroutine accessed by the commercial robot's software. The first controller will be a supervisor which modifies the robot's existing controller. Once this approach has been successfully demonstrated, the robot controller will be completely replaced by a modelling and control hierarchy.

References

[1]Soltec, Sol Vista Park, 12977 Arroyo Street, San Fernando, CA 91340-1597.

[2]Accurate Technologies Incorporated (ATI), P.O. Box 1263, Warren, MI 48090-1263.

[3]D. E. Rumelhart G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," **Parallel Distributed Processing**, Vol. 1, D.E. Rumelhart and J.L. McClelland, and the PDP Research Group, ed. Cambridge. MA: MIT Press, 1986.

[4]R.R. Yager, S. Ovchinnikov, R.M. Tong and H.T. Nguyen, **Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh**, Wiley-Interscience 1987.

[5]P.T. Kazlas, P.T. Monsen, and M.J. LeBlanc, "Neural Network-Based Helicopter Gearbox Health Monitoring System," **Neural Networks for Signal Processing III: Proceedings of the 1993 IEEE Workshop**,September 6-9, 1993.

[6]M. Ulieru, "Diagnosis by Approximate Reasoning on Dynamic Fuzzy Fault Trees," **ICNN '94: Proceedings of the IEEE International Congress on Neural Networks,** June 28-July 2, 1994.

[7]A. Tascillo, "Neural Redundant Robotic Trajectory Optimization with Diagnostic Motor Control," **ICNN '94: Proceedings of the IEEE International Congress on Neural Networks**, June 28-July 2, 1994.

[8]H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-Error-Learning Neural Network for Trajectory Control of a Robotic Manipulator," **Neural Networks**, Vol. 1, pp. 251-265, 1988, pp. 251-265.

[9] B. Kosko, **Neural Networks and Fuzzy Systems**, Englewood Cliffs, N.J.:Prentice Hall 1992.

[10]A. Tascillo, V. Skormin, and N. Bourbakis, "Neurofuzzy Grasp Control of a Robotic Hand," **Neural Networks for Signal Processing III: Proceedings of the 1993 IEEE Workshop**,September 6-9, 1993.

[11]H. Demuth and M. Beale, **Matlab Neural Network Toolbox**, The Mathworks, 1994.

[12]G. Proakis, D. Manolakis, **Introduction to Digital Signal Processing**, MacMillan, New York, 1988.

[13]M. Tascillo, A. Tascillo, and D. Gong, "Neural Network Based System Identification of an Engine Testbed," accepted to IPC '95, Detroit, MI, 1993.

[14]Tascillo, A., and M. Tascillo, "Robot Driver Control with Neural Networks," **Proceedings IPC '95 Workshop**, Detroit, Michigan, May 9-11, 1995, to appear.

[15]Bourbakis, N., M. Maas, A. Tascillo, and C. Vandewinckel, "Odysseus Autonomous Walking Robot: The Leg/Arm Design," **Proceedings AIAA NASA Conference on Intelligent Robotics**, March 1994, Houston, Texas, pp. 29-36.

Appendix A: Fuzzy Logic Applications of Particular Interest

A.1 An Illustrative Application: Obstacle Avoidance

Yochiro Maeda's Fuzzy Obstacle Avoidance Method For a Mobile Robot Based on the Degree of Danger,[1] is a good application to explore how fuzzy logic can be implemented (Figure 1). In this paper fuzzy obstacle avoidance is offered as an alternative to two methods: the Configuration Space Method, which cannot be employed real time due to excessive calculations, and the Artificial Potential Field method, which deadlocks the robot if the target, modelled as a point, falls between the robot and the moving obstacle. It is assumed that only one obstacle is encountered at a time by a robot that can only control its steering, and it only senses continuously moving obstacles ahead of it, not behind it, while facing its fixed target. Robots and obstacles are point masses and it is assumed that the robot responds exactly to the commands it is given.

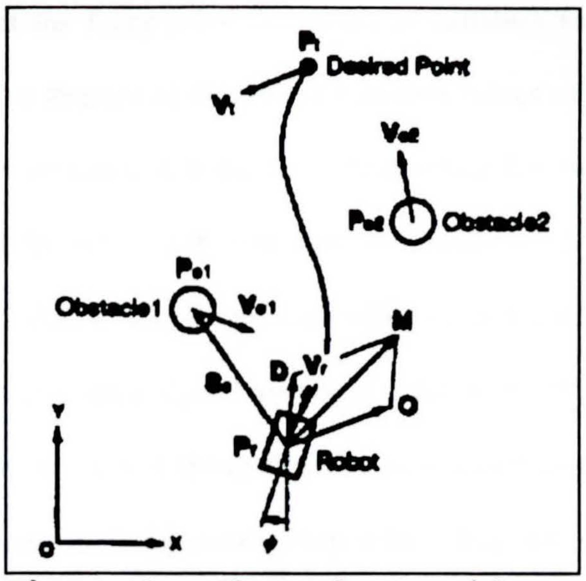


Figure 1: Obstacle avoidance model of a mobile robot.

Degrees of danger for the robot are determined based on the static direction and distance of an obstacle for static danger, α, and velocity magnitude and direction for dynamic danger, β. Figures 2a and 2b show the membership functions for static danger, and Figures 3a and 3b represent those for dynamic danger. Triangular membership functions, used here, are most popular, but other shapes, including those suggested by neural networks after trying hundreds of examples, can also be used.

As an example, if an obstacle is approaching from 30 degrees to the right of the robot's heading and 60 cm away, approximate membership values of 0.7 RS (right small), 0.4 RL (right large), 0.3 N (near), and 0.8 M (medium) would result. If the velocity is approaching from 45 degrees to the right at 50 cm per second, 0.5 VRS (velocity right small), 0.5 VRM (velocity right medium), 0.5 VS (velocity slow), and 0.5 VF (velocity fast) would result. The fuzzy recognition maps of Figures 2d and 3c, fuzzy recognition maps, graphically represent the fuzzy rules that result in variables to be inserted into Figure 2c for their respective degrees of danger. If non-zero values exist for both items in parentheses on the fuzzy recognition map, the corresponding function name, such as RDS for (N) and (RL) on the static map, will hold the minimum of the contributing values. With 0.3 for N and 0.4 for RL, the corresponding value for RDS would be 0.3. Once all relevant function names are assigned membership function values from the fuzzy recognition map, each non-zero membership function is reduced to the area below a vertical slice at the maximum resulting membership value. For the example, the final candidates for static danger would be determined by:

(a) MIN(0.7 RS, 0.3 N) = .3 RDL

(b) MIN(0.7 RS, 0.8 M) = .7 RDS

(c) MIN(0.4 RL, 0.3 N) = .3 RDS

(d) MIN(0.4 RL, 0.8 M) = .4 RSS

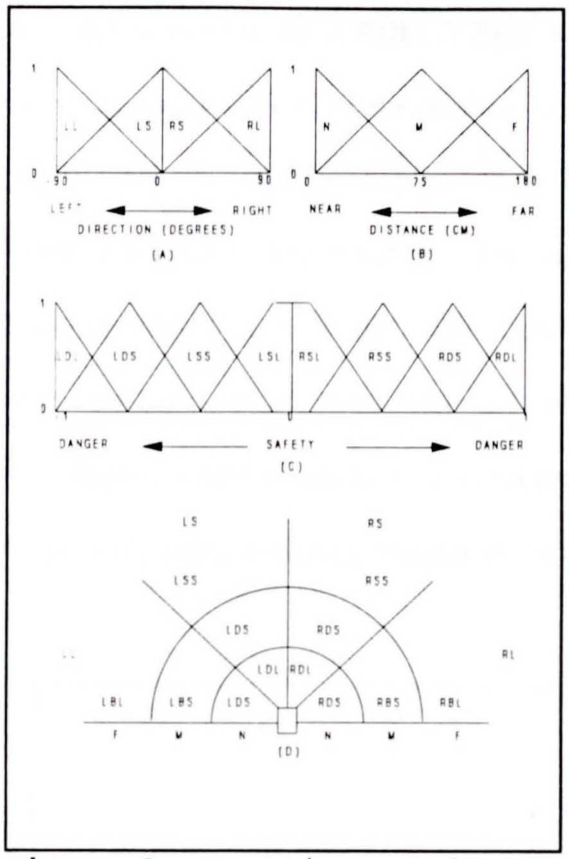


Figure 2: Functions and maps for static danger and degree of safety.

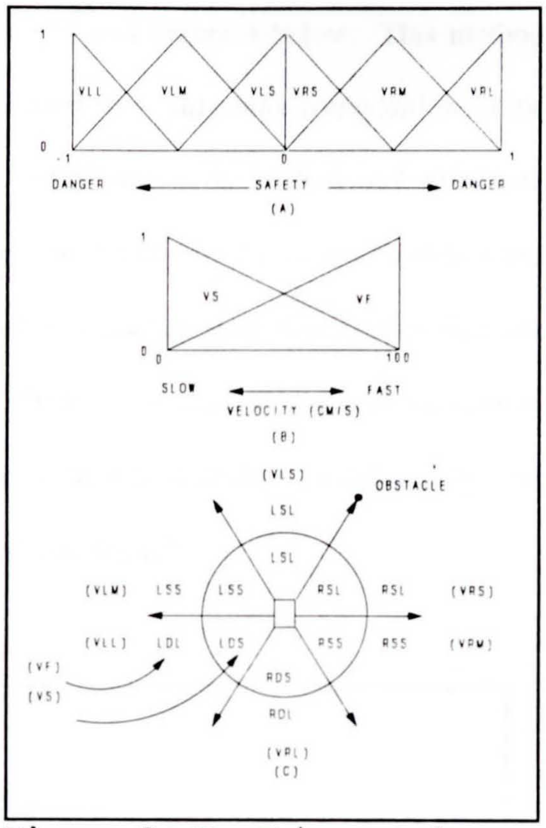


Figure 3: Functions and maps for dynamic danger.

resulting in maximums .3 RDL, .7 RDS, and .4 RSS and Figure 4 below. This method of taking the maximum of the minimums is referred to as min-max composition. One popular way to optimize response time of a fuzzy system is to find and eliminate extraneous rules such as (c) above. For large systems hundreds of possible scenarios are often introduced to a neural network dedicated to isolating only those rules that are necessary for operation. One extreme example, the inverted pendulum discussed later in this chapter, would return to its original orientation despite disturbances using only 7 of 49 possible rules, assuming vertical as the initial condition.[2]

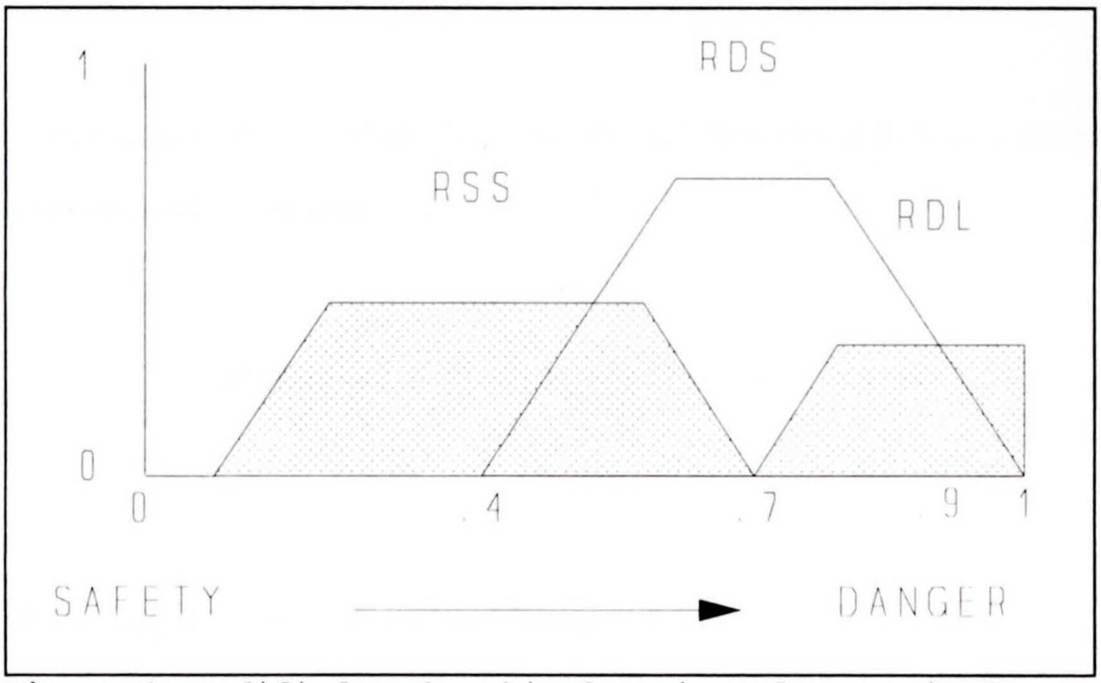


Figure 4: Modified membership functions for static degree of danger.

Next a crisp, or defuzzified, degree of danger needs to be extracted, and this can be accomplished in many ways. Three popular methods are the centroid, or center of gravity, method, the maximum height method, and the averaging method. Centroid is most accurate but time consuming compared to the other two methods, which may be just as suitable for some applications. For the example, the centroid method, by summing across the membership values multiplied by the horizontal center of mass values, and then dividing by the summed membership values, a weighted center of mass will result:

$$\frac{(0.4)(0.4)\ +\ (0.7)(0.7)\ +\ (0.3)(0.9)}{0.4+0.7+0.3} = \frac{0.92}{1.4} = 0.66\ .$$

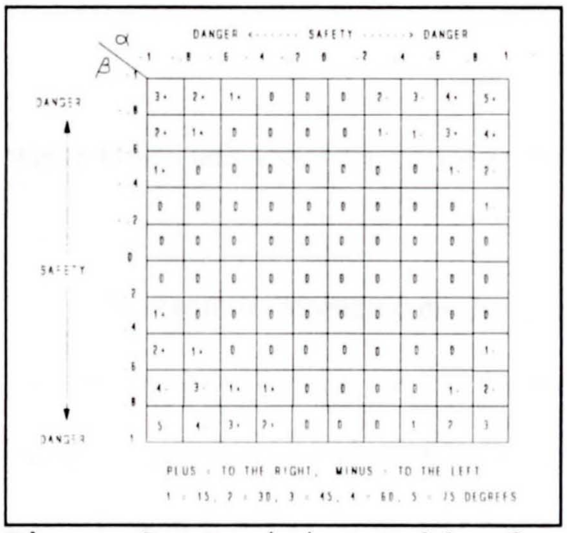
The maximum height method simply declares that 0.7, the center of RDS with maximum membership, 0.7, is the final value:

$$\mathit{MAX}\ (0.4,\ 0.7,\ 0.3) = 0.7 \dashrightarrow 0.7\ ,$$

and averaging the horizontal centers of mass results in:

$$\frac{(0.4 + 0.7 + 0.9)}{3} = 0.67 \ .$$

For this example all three methods result in an α of roughly 0.7, and when combined with a β of 0.25, an avoidance vector of 0.0 can be found with the lookup table of Figure 5. So, for the example, the robot is instructed to maintain its intended direction, which is toward its target. For applications where only one defuzzified value is needed (instead of our α and β), a direct conversion can be made to a control input via a conversion function, circumventing the lookup table.



DANGER <------- SAFETY -------> DANGER

β \ α	-1 – -.8	-.8 – -.6	-.6 – -.4	-.4 – -.2	-.2 – 0	0 – .2	.2 – .4	.4 – .6	.6 – .8	.8 – 1
DANGER -1 – -.8	3+	2+	1+	0	0	0	2-	3-	4+	5+
-.8 – -.6	2+	1+	0	0	0	0	1-	1-	3+	4+
-.6 – -.4	1+	0	0	0	0	0	0	0	1-	2-
-.4 – -.2	0	0	0	0	0	0	0	0	0	1-
-.2 – 0	0	0	0	0	0	0	0	0	0	0
SAFETY 0 – .2	0	0	0	0	0	0	0	0	0	0
.2 – .4	1+	0	0	0	0	0	0	0	0	0
.4 – .6	2+	1+	0	0	0	0	0	0	0	1-
.6 – .8	4-	3-	1+	1+	0	0	0	0	1-	2-
DANGER .8 – 1	5	4	3+	2+	0	0	0	1	2	3

PLUS = TO THE RIGHT, MINUS = TO THE LEFT

1 = 15, 2 = 30, 3 = 45, 4 = 60, 5 = 75 DEGREES

Figure 5: Decision table for the avoidance vector.

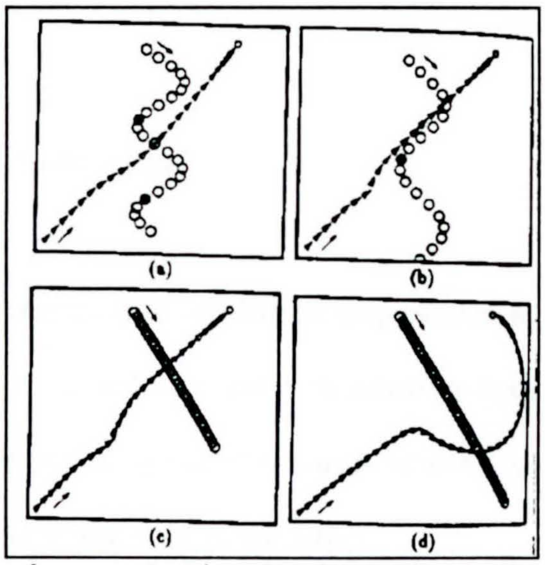


Figure 6:Simulation results.

In order to move the robot, a new target vector, D, is found,

$$D = \frac{P_t - P_r}{| P_t - P_r |} , \qquad (1)$$

where P_t is position of the target and P_r is the position of the robot, shown in Figure 1. Our final goal, the steering vector, M, is found,

$$M = \frac{D + O}{| D + O |} , \qquad (2)$$

where O is a unit vector translation of the avoidance vector.

In Figure 6, depending on the speed of approach of an obstacle (represented by the large circles), slow (a), or fast (b), the robot reached its target with relatively little change of direction. When the fuzzy (c) method was compared to the artificial potential field (d) method, the fuzzy method found a more direct route to the target.

After implementing her own version of this method, the author of this effort found that one advantage to this approach is that the robot always moves, avoiding the possibility of stalling. On the other hand, if the object makes a lot of complicated movements, the object seems to oscillate a unit in either direction. This does not seem apparent in their simulation illustrations, but then the object of interest seems to always be moving either parallel or perpendicular to the robot. If the author were to lessen the sensitivity of her robot, such oscillations might go away.

A.2 The Fuzzy Advantage For Pole Balancing

This apparently simple and natural method of reasoning has proven successful with systems at the limits of and beyond the capabilities of classical control. Although balancing control of an inverted pendulum on a monorail cart can be determined by classical dynamics,[3] the smallest parameter errors in implementation can easily jeopardize success. The actual transition point between balance and instability can be found by employing fuzzy inference rules. If the cart is currently travelling with velocity v at a distance x from a target location, fuzzy rules calculate a virtual equilibrium point for the pendulum that leans away from the target θ degrees. The cart then moves to achieve this virtual equilibrium, tipping the pendulum roughly θ degrees toward the target as a result. Now a second set of rules commands the cart to move toward the origin to drive the pendulum back to the actual equilibrium at vertical. Figure 7 represents the resulting control system. If the cart is to follow some reference path instead of holding one position, the position feedback is replaced by an error in position between the desired and current positions, requiring a control system similar to that in Figure 7. Due to the integral nature of the added inference block, a step disturbance will exist, causing both the cart and pendulum to waver a small distance from their desired locations.

Inputs to a fuzzy controller are a fraction of a maximum value determined by a non-zero winning membership function (or overlapping of functions), resulting in robust solutions which require little beyond rescaling or normalizing. There is no possibility of a divide by zero or other discontinuity. For a pendulum, parameters such as the length of the shaft, its weight, or the controlling motor's torque can naturally drift or be varied without having to change the logical rules or membership functions. Neural networks can be employed to modify the shapes of the membership functions for smooth transitions in control response across all possible inputs, or to eliminate some of the rules from the rule base, decreasing calculation time. These features have allowed Takeshi Yamakawa[4] to balance an active live mouse in a basket at the top of the inverted pendulum, or by extending beyond the current ability of classical control, to simultaneously balance three inverted pendula connected in series above the same cart.

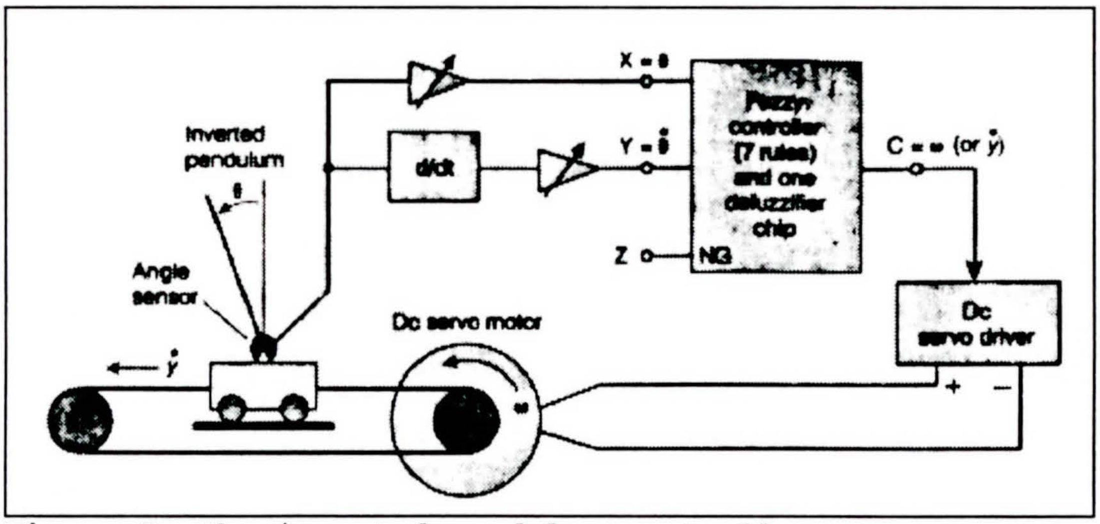


Figure 7: The inverted pendulum controller.

APPENDIX B: FUZZY LOGIC AND NEURAL NETWORKS TOGETHER

B.1 Introduction

Both fuzzy logic and neural approaches are numerical, can be partially described by theorems, and admit an algorithmic characterization that favors silicon and optical implementation.[1] Therefore, they can be interchangeably implemented, and even mixed within the same implementation. Some applications found their way into both questions due the fuzzification of the neuralification of the fuzzification and vice versa. Ideas (mine and others) are presented first, followed by some application examples which are briefly outlined to illustrate concepts, either as implemented or as a possible enhancement to the work published.

B.2 Fuzzification of Neural Networks

If one assumes that a certain imprecision exists or can be tolerated in the controller, then a neural controller can be fuzzified. Then, via verification and deletion, membership function parameters and rules representing only the most crucial input/output relationships can be stored more compactly with a minimum loss of accuracy. The software implementation of fuzzy logic requires fewer multiplies and adds than a neural network, a crucial consideration while neural chip (and chip dedicated software) prices remain high. Some specific applications are described below.

B.2.1 Determining the Shape of a Membership Function

A software fuzzy controller package called Neufuz4[2] uses a modified supervised neural net with error back propagation to determine appropriate nonlinear bellshaped membership functions (for better fit and therefore fewer rules), which is described in better detail for the next question. This neural net, for more automated implementation, is constrained to four inputs and one output, although multiple nets can be used for multiple outputs. The software package provides utilities for aiding the user in gathering training set input/output patterns if they are directly available from the target application, but leaves it up to the user to guess appropriate inner and output layer learning rates as well as final target error. Accumulated knowledge could be clustered and rules generated to suggest appropriate first guesses for the class of problems a given user sees in his work. It is also recommended by the software manufacturer that the user specify the maximum number of membership functions (seven) at first, and then reduce these until the resulting error becomes unacceptable. This process could be fuzzified in a similar manner.

B.2.2 Product Space Clustering

For little extra computational cost one can generate a set of structured FAM rules to approximate a neural system's behavior[1] and speed up computation. Unsupervised neural clustering algorithms efficiently track the density of input-output samples in fuzzy associative memory (FAM) cells. The number of synaptic vectors in each FAM cell can be used to weight the underlying expert or process rules. The fuzzy system can then be tuned by refining the rules with common sense and with further training data. Kosko's applications apply the adaptive product-space clustering methodology to inverted-pendulum control, backing up a truck-and-trailer in a parking lot, and real-time target tracking.

B.2.3 Stability Enhancement by Interpreting the Outputs of an Online Adaptive Neural Controller

In order to solve the inverse kinematics problem for a robot (modelled to mimic the human arm), the usual method of constructing the controller neural network with sigmoidal activations leading to linear activations in the final output layer is replaced by a series of smaller sigmoid-only networks leading into a selector which minimizes the residual error online via a Newton-Raphson iterative method.[3] This selector layer could easily be fuzzified, as the iterative method assumes a solution without local minima between initial and final conditions. Building upon experience, a fuzzy layer could add structure and direction in an online application such as this where stability is crucial.

B.2.4 Distributing the Influence of Statistical Activation Functions

A variant on the above approaches has been advocated by the Bayesian community. Given an MRI brain image application, where background, scalp, skull, white matter, and grey matter must be determined, a statistical classifier has been used to map the original image pixel grey levels onto an *n-tuple of probabilities*[4] representing the likelihood that a pixel belongs to a certain class. The EM (Expectation Maximization) algorithm[5] is used to iteratively determine the proper weighting of Gaussian functions to represent this classification using sample data. The $\omega_{i,j,a}$ weights

$$\omega_{i,j,a} = \frac{\pi_{\vec{\Delta}_{;a;a}} \, p_{\vec{\Delta}_{;a;a}}(i,j)}{p(i,j)} \qquad \textbf{(1)}$$

used in the update equations are said to act as fuzzy classifications of the data points, where π is the a priori probability that a pixel belongs to a certain class and p is the class's conditional intensity distribution. Deltas indicate values for a pair of pixels under consideration. The probabilistic representation as inputs to the classifier led to greater classification accuracy over that of simple grey level inputs. It is assumed that EM adaptation will render the classification process invariant to system noise and shifts in image orientation. This method assumes a priori values of π and initial guesses of intensity parameters for the EM algorithm to modify, requiring human intervention similar to that required to eliminate residual error after an initial implementation of classical fuzzy logic. The entire process is represented as a one hidden unit layer network whose classification layer resembles a fuzzy layer combining the output of Bayesian activation functions. The authors of this paper point out a similar effort by Perlovsky[6] where the EM algorithm is used in an adaptive neural architecture where the fuzzy classification values are seen as adjustable weights between processing elements.

B.2.5 Diagnostic Robotic and DC Motor Control

A neural network can be used to learn the output it needs from a preliminary fuzzy controller to provide the necessary input to a controlled plant in order to obtain the desired response. The fuzzy controller can continue to exist as a backup robust controller if no adequate neural network has been trained for that operating region.

Neural networks can be used to weight contributions particular redundant robot joints can make to satisfy a particular reference command. Fuzzy logic can determine from experience what initial weights to start with at the beginning of a session before enough data has accumulated for diagnostics.

The robot joint controller network has three diagnostic outputs which give their estimations of parameter drift. If two parameters are drifting at the same time, or if there are interdependencies, a fuzzy interpreter can decide which pretrained child neural network is better to drift to, one advocated by diagnostic output A (i.e., stiffness) or by diagnostic output B (i.e. inertia). Because I have demonstrated through examples that a gradient shift in control from a parent network to a child network is stable, then a fuzzy linear allocation of control signals from various networks might provide a most accurate control.

B.3 Neural Networks to Aid Fuzzy Logic

It is difficult to prove absolute stability for a neural net controller that adapts online beyond those with only one layer of adjustable weights with a linear activation function at the output. A fuzzy controller, by having explicit reactions that are clearly outlined for the user, is more accepted, especially if adaptation simply mixes a group of otherwise stable fuzzy outputs. Further, many control problems require nulling a scalar error measure and maintaining that equilibrium state, which may not be modelled exactly by a neural network concerned with representing major nonlinearities. Fuzzy logic can explicitly provide an equilibrium action. Adaptive fuzzy systems can use neural (or statistical) techniques to abstract fuzzy principles from sampled cases and to gradually refine those principles as the system samples new cases.

Neural networks (especially in hardware) can help alleviate several disadvantages of fuzzy logic for control. First, fuzzy logic requires a good conceptual understanding of the relationship between the input and output. Neural networks can give a designer a good rough estimate of the variable dependencies in a given control problem, so that he will be able to focus his efforts more intelligently when designing a fuzzy controller. A neural network that is provided with the best possible inputs (or time dependencies of inputs) will not train as well as one with the inputs most influential to the relationship.

Secondly, analytic controllers generally have a more rapid response time, and thirdly, subsets of rules must be defined continuously over all regions of each universe of discourse or discontinuous control actions may occur. If a neural network can provide a better fit for each rule (i.e. by finding the best bell curve shape), and clustering (i.e. fuzzy C-means[7] or adaptive product-space[1]) techniques prune out redundant or unnecessary rules, then the fuzzy controller can act that much faster, and can compensate for any remaining speed discrepancy with more robustness than its analytic counterpart.

Fourthly, membership calculations grow exponentially with the universe of discourse. If the task becomes too complex for control at one level, a hierarchy with trainable connections between levels can capture wider reaching trends and allow the lower levels to focus on local relationships.

Fifthly, fuzzy logic requires complete knowledge of sensor/actuator relationships. Neural networks can extract coupled effects from a missing input by assuming a similar relationship with what it found in training data. Once the relationship is found for a neural controller, for instance, the diagnostic outputs on the same network can be trained with only the added expense of connections from the last hidden layer. One successful combination could be a hybrid system where neural networks handle relationships with unreliable or missing inputs and fuzzy quickly calculates other relationships.

Finally, one must verify that a system originally controlled via a neural network still results in a stable plant when converted to one that is fuzzy.

Kosko[1] states that neural networks so far seem best applied to ill-defined two-class pattern recognition problems. For a one hidden layer network, this is probably so. Higher order problems are being successfully learned with more hidden unit layers, however, and the previous problems are being learned more efficiently than with more hidden units in one layer. Fuzzy logic will be able to capitalize on this increased ability to learn nonlinear relationships.

Some nonlinear relationships are difficult for humans to estimate from input output data alone. A highly nonlinear hydrogen etch process with a very small acceptable process window caused engineers at IBM East Fishkill to struggle for weeks adjusting the two variables, pressure and gas flow. Some specific applications are described below.

B.3.1 Estimation, Storage, and Modification of Fuzzy Rules

Kosko allows competitive neural networks to adaptively estimate, store, and modify decomposed FAM rules in a matrix format.[1] This representation requires far less storage than the multidimensional-array representation.

B.3.2 Estimation of Inaccessible Controlled Plant States

A neural network can act as a state estimator to feed a fuzzy controller with variables that are not directly accessible. Neural network structure and weight update methods provides a convenient way to combine the outputs of the various fuzzy logic rules, without having to know the exact structure of the function(s) to be learned. The membership functions themselves can be provided by the domain experts and remain as static information, or can be automatically generated using some statistical methods such as Set-Valued statistics[8], or to be trained using a neural network and based on historical data from the data base.

B.3.3 Refinement of the Neural Widrow Truck and Trailer Emulator

Because the fuzzy clustered modified neural and fuzzy truck systems generated similar backing-up trajectories, black-box neural estimators seem adequate for defining the front end of FAM-structured systems[1].

B.3.4 Learning the Constants for Neural Network Training

Neufuz4, the software package mentioned earlier, executes the final fuzzy control in a standard fashion: it first finds each rule that applies to inputs at hand, calculates the degree of membership for inputs for each rule, and finally multiplies the memberships for all inputs. In order to determine an accurate, compact fuzzy controller, a neural network approach is used to define the shape of the membership functions, and a final weighting layer of these functions can be trained to provide an optimal combination to suggest as an output.

The first layer neurons and the weights between layer 1 (inputs) and layer 2 (rules) are used to define the input membership functions. The rule base is reflected in layer 2, where the inputs to one of the neurons are the antecedents and the output is the consequent for a single fuzzy rule. If input 1 is low and input 2 is medium then output is (crisp) X. All of these X's are multiplied, then defuzzified at the output layer.

To simplify implementation fuzzy rules are limited to use of the AND operator (corresponding to the conventional fuzzy *min* operation) and output is always a singleton. To increase the effectiveness of the implementation and to be able to transfer assembly code to a low cost 8 bit microcontroller, inputs to the controller are restricted to four and shouldered trapezoids are approximated onto the calculated bell curve shapes. The number of rule (hidden unit layer) neurons is equivalent to the maximum number of rules:

$$\text{rules} = (\text{number of membership functions/input})^{\text{inputs}}.$$

Caution should be used when combining the efforts of enhanced fuzzy logic, via neural networks or other approaches, as their output characteristics may differ and complicate tuning of a final result. For example, because Neufuz4 first uses a neural network to determine the shape of a gaussian membership function, and then approximates it as a shouldered trapezoid (to save ROM), the highest possible membership may not be one. Therefore, a maximum possible output signal may never be achieved, and overlapping memberships may not add to one for a given input value.

B.3.5 Higher Level Control Strategy

When considering a higher level of control (i.e., which fuzzy logic path to follow for navigation and obstacle avoidance[9]), neural networks can be used for online tuning of decision threshold and gain limits. A conservative navigator can slowly shorten distances as experience confirms an adequate margin of error after each safety threatening situation is encountered.

B.3.6 Fuzzy Control With Feedback Error Learning

No exact kinematics or dynamics model is needed for the fuzzy control of a simulated biped locomotive robot when estimated torque feedback is used for neural tuning of the defuzzification weights[10]. For this application fuzzy logic learns the nonlinear relationship necessary for a smooth gait when only "head" and "ankle" reference trajectories are given. Continuously differentiable gaussian and quadratic membership functions are used in antecedent parts of fuzzy rules in order to obtain a continuous error signal. In order to minimize feedback torque, and thereby smooth the gait, the learning rule becomes:

$$w_j[n+1] = w_j[n] + \tau * T_b[n] * \mu_j[n-10]$$

where w_j is the 5x1 weight vector leaving the jth rule, n is the sample number (every 2 msec), T_b is the 5x1 vector of joint feedback torques, τ is the learning rate, and $\mu_j[n-10]$ is the compatibility degree of the jth rule 10 steps previous.

B.3.7 Fuzzy Gain Scheduling

For a helicopter controller[11], control is dependent upon flying conditions such as height above the ground and wind velocity. After training bidirectional associative memory fuzzy controllers via human experience, a Widrow-Hoff Algorithm[12] is used to refine each operation model under its flying condition. Fuzzy allocation can then be used to choose a combination of controllers for ambiguous conditions.

B.3.8 Neural Nets to Store Fuzzy Estimations

Separate storage of fuzzy rule associations consumes space but provides an audit trail of the fuzzy rule inference procedure and avoids crosstalk. The user can add endpoint rules or delete unnecessary rules without disturbing the others and neural networks can estimate or individually house the estimations.

References

[1]Kosko, B., **Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence**, Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[2]Thaler, Stephen, "Fuzzy rule generation based on a neural network approach," **Electronic Engineering**, p. 43-50, July 1993.

[3]Guez, A., Z. Ahmad, and J. Selinsky, "The Application of Neural Networks to Robotics," **Neural Networks: Current Applications,** P. Lisboa, ed., New York: Chapman and Hall, p. 111-122, 1992.

[4]Toulson, D.L., J.F. Boyce, C. Hinton, "Data Representation and Generalization In An Application Of a Feed-Forward Neural Net," **Neural Network Applications**, J. G. Taylor, ed., New York: Springer-Verlag, 1992.

[5]Dempster, A.P., N.M. Laird, and D.B. Rubin, "Maximum Likelihood Estimation From Incomplete Data via the EM Algorithm (with discussion), **J.R. Statis. Society**, B 39, p. 1-38, 1977.

[6]Perlovsky, L.I., and M.M. McManus, "Maximum Likelihood Neural Networks for Sensor Fusion and Adaptive Classification," **Neural Networks**, 4-2, p. 89-102, 1991.

[7]Pedrycz, Witold, "Design of Fuzzy Control Algorithms with the Aid of Fuzzy Models," **Industrial Applications of Fuzzy Control**, M. Sugeno, ed., North-Holland: Elsevier Science Publishers, 1985.

[8]Wang Pei-zhuang, "Random sets in Fuzzy Set Theory," **Systems and Control Encyclopedia**, Madan G. Singh, ed., Pergamon Press, pp. 3945-3947, 1988.

[9]Maeda, Y., M. Tanabe, Y. Morikazu, and T. Tomohiro, "Control Purpose Oriented Behavior-Decision Fuzzy Algorithm with Tuning Function of Fuzzy Branch," **Fuzzy Engineering Toward Human Friendly Systems**, IFES '91, Part VII, Fuzzy Logic Control, p. 694-705, 1991.

[10]Ashida, H., and H. Ichihashi, "Fuzzy Learning Control of a Biped Locomotive Robot," **Fuzzy Engineering Toward Human Friendly Systems**, IFES '91, Part VIII, Application, p. 1013-1023, 1991.

[11]Yamaguchi, T., K. Goto, M. Yoshida, Y. Mizoguchi, and T. Mita, "Fuzzy Associative Memory System and Its Application to a Helicopter Control," **Fuzzy Engineering Toward Human Friendly Systems**, IFES '91, Part VII, Fuzzy Logic Control, p. 770-779, 1991.

[12]Widrow, B., and M. Hoff, "Adaptive Switching Circuits," **WESCON Conference Proceedings**, Volume 4, 1960.

APPENDIX C:

THE BACKPROPAGATION ALGORITHM

Each input to a node of a neural network is entered into an activation function after summing with other inputs and continuing on to the next node. An activation function is desirable so that the nodal output is continuously differentiable, has a simple, easy to compute derivative, and can give a reasonable response to inputs with extreme values. Common examples vary from simple linear or threshold relationships to sigmoidal functions such as:

$$f = \tanh((\Sigma input_i w_{ij}) + b_j) \qquad \textbf{(1)}$$

and

$$f = \frac{1}{1+e^{-\alpha((\Sigma input_i wij)+b_j)}} \qquad \textbf{(2)}$$

Alpha, α, is a constant that, when increased, increases the slope of the sigmoid curve of the activation function. An α of three is already quite vertical. When calculating weight corrections, the derivative of the activation function must be evaluated at that node. Derivatives of the above two functions are:

$$f' = \frac{1}{(\cosh(\Sigma\, input_i w_{ij}) + b_j)^2} \tag{3}$$

and

$$f' = \frac{\alpha e^{-\alpha((\Sigma\, input_i w_{ij}) + b_j)}}{1 + e^{-\alpha((\Sigma\, input_i w_{ij}) + b_j)}} \tag{4}$$

Noting that the activation function corresponding to this last derivative is quite similar to the solution for the spread of epidemics from differential equation textbooks, a much simpler derivative is often employed:

$$f' = \alpha\, f\, (\, 1 - f\,) \tag{5}$$

where f is the percentage of infected members, and 1-f the percentage of non-infected members.

For each training set example, $tset_i$, deltas δ, or incremental changes, can be calculated for both biases and weights. For a final output node, find δ_k by subtracting the calculated activation value from the desired value and multiplying by the derivative of the activation function at that node:

$$\delta_k = (\Sigma(\bar{f}_k - f_k))\ f'_k\ ((\Sigma input_j w_{jk}) + b_k)\ , \tag{6}$$

where $\bar{f}_k$ is the desired activation value and f_k is the calculated activation value. For a hidden unit node, δ_j is calculated by first summing the associated input deltas multiplied by the weights connecting their output nodes, then multiplying the sum by the derivative of the activation function at that node as such:

$$\delta_j = (\Sigma(\delta_k w_{jk}))\ f'_j\ ((\Sigma input_i w_{ij}) + b_j)\ . \tag{7}$$

Now one can finally update biases and weights before introducing the next training set example. The change in bias weight (referred to as b, since the actual value of the bias is always one) of an output or hidden value node is calculated by first summing its associated deltas, then multiplying by eta, η, a learning rate either specified by the user or keyed to the complexity of the error surface to keep the net from neither stalling nor going unstable :

$$\Delta b_k = \eta \Sigma \delta_k\ , \tag{8}$$

and

$$\Delta b_j = \eta \Sigma \delta_j\ . \tag{9}$$

The change in weight value for an output or hidden unit node is obtained by first summing the products of each associated delta multiplied by each associated activation function, then finally multiplying by eta:

$$\Delta w_{jk} = \eta \Sigma \delta_k f_j \ , \tag{10}$$

and

$$\Delta w_{ij} = \eta \Sigma \delta_j f_i \ . \tag{11}$$

```
        4) Find changes in connection w's, given learning rate η & momentum factor κ:

$$\delta w_{ij} = \kappa * \eta * f_i * f_j * \left(\frac{\delta \epsilon}{\delta f}\right)_j \qquad (14)$$

        }End Input Loop

}End Tolerance Loop
```