PERGAMON

Computers and Electrical Engineering 26 (2000) 5–32

Computers and Electrical Engineering

www.elsevier.com/locate/compeleceng

Soft comuting for autonomous robotic systems[☆]

M.-R. Akbarzadeh-T*, K. Kumbla[1], E. Tunstel[2], M. Jamshidi

Center for Autonomous Control Engineering, 125 EECE Building, University of New Mexico, Albuquerque, NM 87131, USA

Received 1 May 1997; received in revised form 1 March 1999; accepted 1 April 1999

Abstract

Neural networks (NN), genetic algorithms (GA), and genetic programming (GP) are augmented with fuzzy logic-based schemes to enhance artificial intelligence of automated systems. Such hybrid combinations exhibit added reasoning, adaptation, and learning ability. In this expository article, three dominant hybrid approaches to intelligent control are experimentally applied to address various robotic control issues which are currently under investigation at the NASA Center for Autonomous Control Engineering. The hybrid controllers consist of a hierarchical NN-fuzzy controller applied to a direct drive motor, a GA-fuzzy hierarchical controller applied to position control of a flexible robot link, and a GP-fuzzy behavior based controller applied to a mobile robot navigation task. Various strong characteristics of each of these hybrid combinations are discussed and utilized in these control architectures. The NN-fuzzy architecture takes advantage of NN for handling complex data patterns, the GA-fuzzy architecture utilizes the ability of GA to optimize parameters of membership functions for improved system response, and the GP-fuzzy architecture utilizes the symbolic manipulation capability of GP to evolve fuzzy rule-sets. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Soft computing; Neural networks; Genetic algorithms; Genetic programming; Fuzzy logic; Robotic control; Artificial intelligence

[☆] This work was supported in part by NASA contract ♯NCCW-0087 and by a Waste-management Education and Research Consortium grant under Award ♯WERC/NMSU/DOE Amd 35.

* Corresponding author.

E-mail address: akbazar@eece.unm.edu; http://ace.unm.edu (M.R. Akbarzadeh-T).

[1] Currently with IBM Corporation, San Jose, CA, USA

[2] Currently with Jet Propulsion Laboratory, Pasadena, CA, USA

1. Introduction

Traditional robot control methods rely upon strong mathematical modeling, analysis, and synthesis. The approaches, proposed to date, are suitable for control of industrial robots and automatic guided vehicles which operate in structured environments and perform simple repetitive tasks that require only end-effector positioning or motion along fixed paths. However, operations in unstructured environments, such as in remote planets and harzardous waste sites, require robots to perform more complex tasks without an adequate analytical model. In cases where models are available, it is questionable whether or not uncertainty and imprecision are sufficiently accounted for. For example, robot manipulators in waste handling applications operate in the unstructured and unknown environment of a waste tank or a waste disposal site. Such robots may be required to repair a leak for an underground aging tank, or to manipulate the hazardous waste inside the tank which may have been deposited and solidified over time. This concept also applies to space applications where robots must autonomously, and with little or no communication with Earth, operate in harsh environments and perform complicated multi-faceted tasks.

The advent of fuzzy logic provided a powerful tool for control of such mathematically complex, uncertain or noisy systems [1,2]. Fuzzy logic control offers an attractive alternative to the existing highly mathematical conventional controllers proposed for such complex systems. Herein, we refer to a complex system as one for which model-based classical control techniques cannot easily render a reasonable solution due to ambiguities, uncertainties and/or nonlinearities in the complex system's mathematical model. Fuzzy controllers are robust in the presence of perturbations, easy to design and implement, and efficient for systems that deal with continuous variables [3].

The application of fuzzy logic relaxes the need for an accurate mathematical model of the system by replacing the mathematical knowledge with human (expert) knowledge and intuition. As a result, performance of a fuzzy logic controller is a function of the quality of its embedded expert knowledge rather than a highly accurate mathematical model. Developing quality expert knowledge, however, can be a time-consuming and costly endeavor. Furthermore, a human expert may find it difficult to express his/her control actions, which are often partly decided at a subconscious level, in terms of a set of constrained rules and membership functions. Also, in most instances, the available knowledge-based controller may adequately control a given process, but may not necessarily be the optimal controller.

The process of knowledge acquisition is indeed a challenging problem in fuzzy logic; and up until now, there is not yet a systematic method for knowledge acquisition in conventional applications of fuzzy logic control. Consequently, in many practical instances, fuzzy control alone is not sufficient for addressing the complex intelligent control problems of robotic systems. Additional tools are necessary to achieve adaptation and learning capabilities. The control schemes described herein are examples of approaches that augment fuzzy logic with other *soft computing* paradigms [4] such as neural networks (NN), genetic algorithms (GA), and genetic programming (GP) to achieve the level of intelligence required of complex robotic systems.

In this expository paper, three dominant hybrid fuzzy control approaches are applied to address various robotic control issues which are currently under investigation at the NASA

Center for Autonomous Control Engineering. The first methodology uses neural networks to learn rules and change membership functions for a fuzzy logic controller (FLC) of a direct drive motor, used in many industrial robots. The second methodology develops a two-level hierarchical fuzzy control structure for flexible manipulators. It incorporates genetic algorithms [5] in a learning scheme to adapt to various environmental influences on the system. The third approach incorporates fuzzy logic control, with rule learning by genetic programming [6], into the framework of behavior-based control for mobile robot navigation. Experimental results of fuzzy controllers learned with the aid of these soft computing paradigms are presented.

2. Real-time neuro-fuzzy control

This section describes a novel architecture for intelligent control of robotic systems by combining the capabilities of NN and fuzzy logic paradigms. In particular, a neuro-fuzzy controller is presented which uses NN to modify the parameters of an adaptive fuzzy logic controller. The adaptability of the fuzzy controller is derived from a *rule generation mechanism* and modification of the scaling factor or the shape of the membership functions. The *rule generation mechanism* monitors the system response over a period of time to evaluate new fuzzy rules. Non-redundant rules are appended to the existing rule base during tuning cycles. The membership functions of the input variables are adjusted by a *scaling mechanism*. A multi-layer perceptron neural network classifies the temporal response of the system into various patterns such as oscillatory behavior, overshoot in response, or steady state error, etc. This information is used by the decision mechanism which determines the scaling factor of the input membership functions. Another neural network identifies the dynamic system, hence acting as a reference model. This model can be used to determine the stability of the new rules generated before applying to the real system.

In order to implement this hybrid controller in real time, it is necessary to have substantial computing power. The TMS320C30 digital signal processor from Texas Instruments, with its powerful instruction set, high-speed number crunching capability, and its innovative architecture is ideally suited for such an application [7]. There are commercially available boards based on TMS320C30 chips, which can be installed on a personal computer (PC). A board from DSP Research has been utilized for this purpose [8]. The software for the control algorithm is developed in C-language and is compiled and downloaded to the DSP board. Collectively, these computing resources are used to implement the neuro-fuzzy controller architecture in real-time to control a direct drive motor used as a robot actuator.

2.1. Neuro-fuzzy controller

The proposed neuro-fuzzy controller uses the capability of a fuzzy logic based controller to control systems, without prior knowledge of the dynamic characteristics, that are nonlinear in nature and for which a mathematical model is unavailable. So there is a need for a nonlinear fuzzy controller to adapt to the changes in model parameters, operating conditions, etc. This requires a mechanism by which the information about these changes can be gathered and used for adapting the fuzzy controller. The extraction of abstract system knowledge is performed by neural networks and their learning capability used in designing the fuzzy controller. This makes the controller robust and *intelligent*, aware of changing environmental and physical conditions. As mentioned above, the scheme uses two neural networks and an adaptive fuzzy controller (see Fig. 1). Each NN has a specific role. One is used for pattern recognition and the other for system identification. The aim of the system is to automatically form the fuzzy controller and tune itself to changes in the operating conditions. During tuning, the adaptive mechanism alters the fuzzy rule base, and scales membership functions to make the system perform in a desired manner.

The response of the system is determined by monitoring the controlled variable for some disturbance. The disturbance can be a step change in the desired input signal. The dynamics of the object system behave in a certain way depending on the present status of the controller. This behavior over a certain period of time (tuning cycle) is monitored. Based on the temporal response, the *fuzzy rule generation algorithm* decides whether a new fuzzy rule has to be formed or not and the *fuzzy scaling mechanism* decides whether a tuning operation is needed or not for the scaling of membership functions associated with the currently established rules. If there are no rules at the initial state of the fuzzy controller, then the algorithm sets up some initial control value U_0 to control the object system.

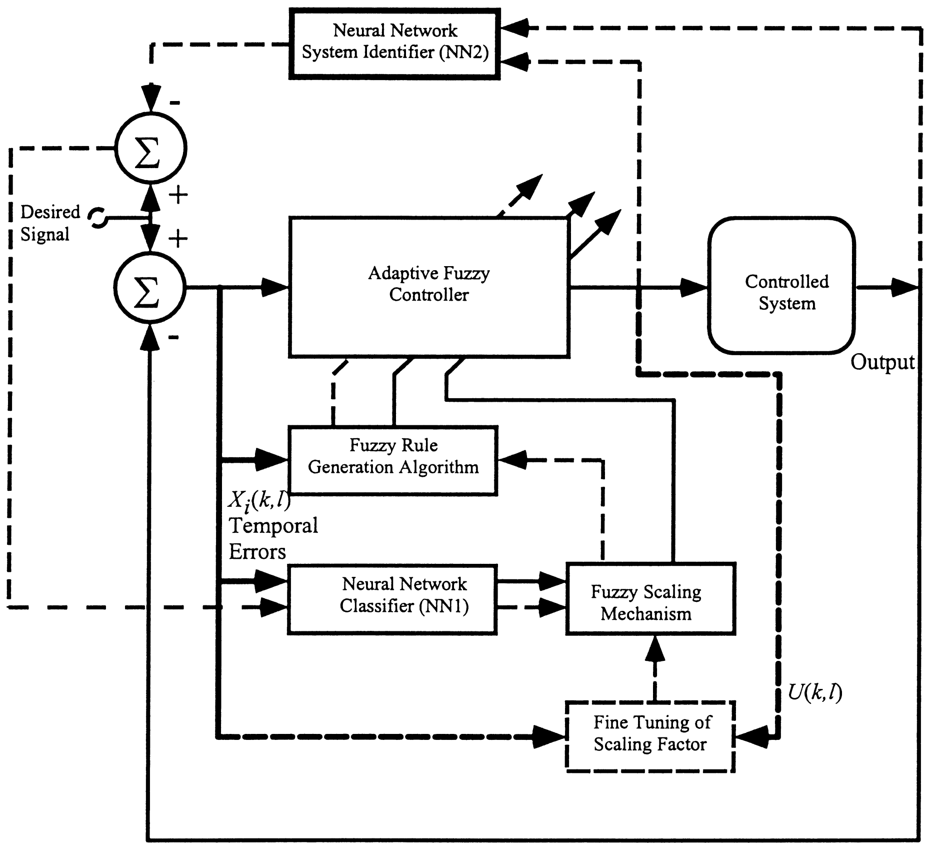

Neural Network System Identifier (NN2)
-
Σ
+
Desired Signal
+
Σ
-
Adaptive Fuzzy Controller
Controlled System
Output
Fuzzy Rule Generation Algorithm
$X_i(k,l)$ Temporal Errors
Neural Network Classifier (NN1)
Fuzzy Scaling Mechanism
$U(k,l)$
Fine Tuning of Scaling Factor

Fig. 1. Block diagram of the adaptive neuro-fuzzy controller.

A static neural network (NN1) is used to identify the pattern of response of the controlled system. NN1 classifies the dynamic responses of the object system to the following typical patterns which it is made to learn as a pre-process to system integration.

- *A*-type: a similar pattern to the desired response.
- *B*-type: an oscillating and slowly converging pattern.
- *C*-type: an overshoot pattern.
- *D*-type: a pattern with a steady state deviation value.
- *E*-type: an asymptotic and slowly converging pattern.
- *F*-type: an oscillating and diverging pattern.
- *G*-type: any other type of response.

Initially, a set of responses with the above characteristics is simulated and the neural network is made to learn these standard patterns. This however may not exactly represent the true characteristics of the object system, in which case there is an *on-line* learning option. The operator in this case determines the typical responses and makes the neural network learn these patterns. Since the neural network has learned the standard patterns, learning similar patterns corresponding to the object system takes less time.

A dynamic neural network (NN2) is used to identify the dynamic characteristics of the controlled system. As a pre-process, neural network NN2 learns the dynamic characteristics of the object system through pairs of input and system response. Identification of the system model is performed off-line by collecting the input and output data over a length of time which constitutes a tuning cycle. Once NN2 is sufficiently capable of identifying the dynamics (i.e. it is able to imitate the real object system), then it can be used to simulate the object system in cases where it is too risky to control the object system with an incomplete fuzzy controller. NN2 can also be used to monitor the stability of the adaptive fuzzy controller or to find the normalizing values of the membership functions to control the object system adequately. There are, however, many questions on identifying the real plant. First of all, the plant identified represents the closed-loop dynamic system rather than the real open loop plant. This is because we are using the fuzzy controller in the loop. There is also a question of the training data being sufficiently rich to take into account all the modes of operation. This means that NN2 may not necessarily have learned all the system characteristics if we have operated the object system under only a limited range of operating condition and limited order of disturbances.

In a typical process plant, the tuning of the various control loops is performed by monitoring the response of the controlled variable to some disturbances. The disturbance may be created by a step change in the desired value. The time history of the controlled variable is recorded on a chart. The gains of the conventional controller are tuned by the operator based on the observation of these responses. Suppose the response of the object system is oscillating, then an increase in the derivative gain of the PID control normally reduces the oscillation. If there is an overshoot in the response, decreasing the proportional gain reduces the overshoot. Likewise if there is a steady state error then increasing the integral gain or increasing the proportional gain would cause the response to approach the desired pattern. This practice is continued until each of the control loops in question is optimally tuned. This operation of course takes into account the experience of the operator.

The same procedure is incorporated into the adaptive neuro-fuzzy controller. The NN1 classifies the response into the patterns A through F as described. The value of each of the outputs of NN1 varies from 0 to 1 depending on how best it can classify a particular pattern. A multi-layer perceptron with a large number of input nodes, for example say 100, corresponding to the input pattern buffer, and a certain number of outputs, say 7 (A through G) can be trained to recognize the various patterns described. The output of the neural network is a set of inputs for the *fuzzy scaling mechanism*. The fuzzy rules in the fuzzy scaling mechanism consist of linguistic rules based on tuning a typical process parameter. The output of the fuzzy scaling mechanism is the incremental change in the scaling factor for the *input* and *output* scaling factor of the adaptive fuzzy controller. Fine tuning of these incremental changes is performed by another *fuzzy fine tuning mechanism*. This mechanism of scaling the input/output membership functions for currently established fuzzy control rules is repeated until the dynamic response is classified to A-type response (desired) by the neural network, NN1.

Let the fuzzy control rules in a system be represented as follows:

$$\text{If } x_1(k) \text{ is } \mu_{x_1^{Ni}} \text{ and } x_2(k) \text{ is } \mu_{x_2^{Ni}} \text{ and } x_3(k) \text{ is } \mu_{x_3^{Ni}} \text{ then } u(k) \text{ is } \mu_{u_i^{Ni}} \tag{1}$$

where $x_1(k)$, $x_2(k)$ and $x_3(k)$ are the error, derivative of error and integral of error of the deviation of the dynamic response of the object system from the desired value and its variation with respect to time k, respectively. The symbols $\mu_{x_1}^{Ni}$, $\mu_{x_2}^{Ni}$, $\mu_{x_3}^{Ni}$ and μ_{u}^{Ni} in (1) indicate fuzzy variables represented by membership functions of the inputs $x_i(k)$ and output $u(k)$ variables, respectively. The number of membership functions of each variable is defined by Ni. Here we assume, for simplicity, all the inputs and output variables contain the same Ni membership functions.

The proposed rule generation algorithm takes the time history of the input variable or the temporal input data and generates a set of fuzzy rules which attempts to control the process. The sample input data is taken over a specific tuning cycle. The tuning cycle represents a specific period where a disturbance or change in the set point of the controlled variables is performed. The response to this disturbance effectively shows the performance of the controller. The rule generation algorithm generates new rules based on the response of the previous tuning cycle. These new rules are augmented with the previous rule set. The new rule set may generate a different response in the next tuning cycle and this may generate another set of new rules. This process continues until the rule base stabilizes in order to control a particular system. Each rule in a rule set is assigned some weight depending upon how many times it is used or fired during a specified number of tuning cycles. There may be some rules which are not fired or are used less frequently after the rule base has stabilized. Least frequently used rules are removed from the rule set. The details of the rule generation mechanism can be found in [9].

The fuzzy scaling mechanism rules are represented as follows:

$$\begin{gathered}\text{If } A \text{ is } R_a^{Nj} \text{ and } B \text{ is } R_b^{Nj} \text{ and } C \text{ is } R_c^{Nj} \text{ and } D \text{ is } R_d^{Nj} \text{ and } E \text{ is } R_e^{Nj} \text{ and } F \text{ is } R_f^{Nj} \text{ then} \\ \Delta\alpha_1 \text{ is } I_{\alpha_1}^{Nj} \text{ and } \Delta\alpha_2 \text{ is } I_{\alpha_2}^{Nj} \text{ and } \Delta\alpha_3 \text{ is } I_{\alpha_3}^{Nj} \text{ and } \Delta\beta \text{ is } O_{\beta}^{Nj}\end{gathered} \tag{2}$$

where A, B, C, D, E, and F are the output of NN1 taken as the input to the fuzzy scaling mechanism. $\Delta\alpha_i$ ($i = 1,2,3$) are increments of the scaling factor of the input variables $x_1(k)$, $x_2(k)$ and $x_3(k)$, respectively and $\Delta\beta$ is the increment scaling factor of the output variable $u(k)$.

The associated membership functions of A, B, C, D, E, and F are R_a^{Nj}, R_b^{Nj}, R_c^{Nj}, R_d^{Nj}, R_e^{Nj}, and R_f^{Nj}, respectively. The membership functions of $\Delta\alpha_i$ are $I_{\alpha_i}^{Nj}$, ($i = 1,2,3$) and membership functions of $\Delta\beta$ are O_β^{Nj}. Nj is the number of membership functions which is assumed to be the same for all the variables.

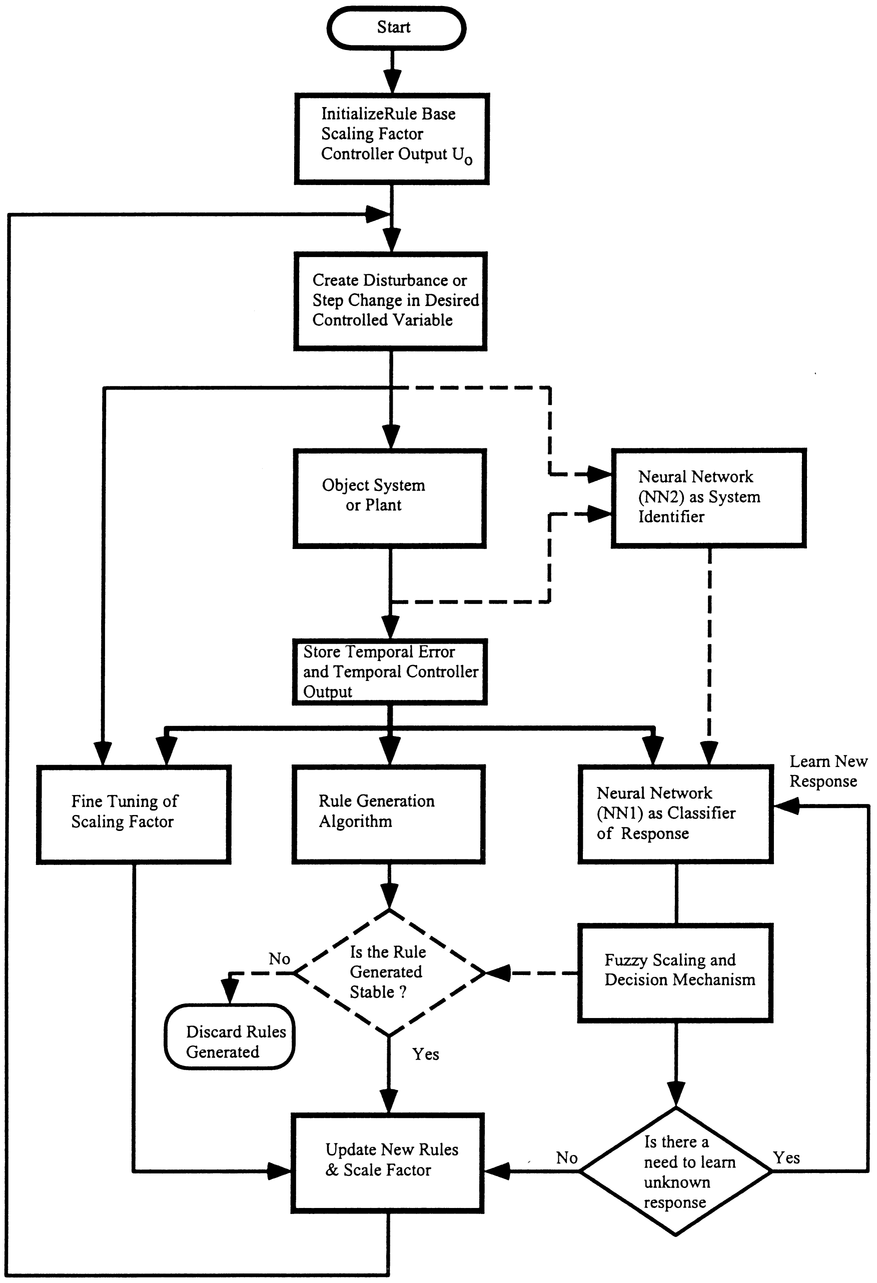

Start
InitializeRule Base Scaling Factor Controller Output U0
Create Disturbance or Step Change in Desired Controlled Variable
Object System or Plant
Neural Network (NN2) as System Identifier
Store Temporal Error and Temporal Controller Output
Fine Tuning of Scaling Factor
Rule Generation Algorithm
Neural Network (NN1) as Classifier of Response
Learn New Response
Is the Rule Generated Stable ?
No
Discard Rules Generated
Yes
Fuzzy Scaling and Decision Mechanism
Update New Rules & Scale Factor
No
Is there a need to learn unknown response
Yes

Fig. 2. Flowchart of the neuro-fuzzy controller.

The above inference results in updating of the scaling factors represented by:

$$\alpha_1(T+1) = \alpha_1(T) + \Delta\alpha_1(T)$$

$$\alpha_2(T+1) = \alpha_2(T) + \Delta\alpha_2(T)$$

$$\alpha_3(T+1) = \alpha_3(T) + \Delta\alpha_3(T)$$

$$\beta(T+1) = \beta(T) + \Delta\beta(T) \tag{3}$$

where T indicates the number of tuning operations.

Fine tuning of the scaling factor may be performed to account for the maximum absolute values of the error during the tuning cycles. The input and output patterns of the object system controlled by the currently established fuzzy control rules are sampled at sampling time k ($k = 0,1,\ldots,l$). The *fine tuning mechanism* then detects the errors $x_i(k)$ ($i = 1,2,3$) and variation $u(k)$ of the input at each sampling time k, and finds the values $x_i(k)^{\max}$ ($i = 1,2,3$) and $u(k)^{\max}$ as follows:

$$x_i^{\max} = \max\{|x_i(k)|\} \quad i = 1, 2, 3$$

$$u(k)^{\max} = \max\{|u(k)|\} \quad k = 0, \ldots, l. \tag{4}$$

2.2. Integration of system components

Fig. 2 shows the flowchart of the neuro-fuzzy controller. The various steps involved in the tuning procedure are as follows.

Step 1 There are no rules in the initial state of the system. The control system sets up the initial constant control value u_0 to make the first fuzzy control rule, and the object system is controlled by this value u_0.
Step 2 Create a disturbance or give a small step change in the desired variable from the initial condition. Perform a tuning cycle.
Step 3 The dynamic responses of the object system controlled by the currently established fuzzy rules (initially U_0) are sampled at sampling time k ($k = 0,1,\ldots,l$) and the errors $x_i(k)$ from the desired value are detected from this sampled information. (For fine tuning $x_i(k)^{\max}$ are then determined.)
Step 4 Send the time history of the response $x_i(k)$ to the rule generating algorithm.
- Create new rules.
- If the new rules are unique, append them to the original rule base. Make sure there are no repeated rules.
- Determine from the weights of the old rules if they should be retained. If not, delete these rules. This is done by determining how many times the rules are fired in the previous tuning cycle.
- If the new rules created are already present, this indicates the rules have stabilized and generation of additional rules is unnecessary.

Step 5 Send the time history of the response $x_i(k)$ to NN1 (the classifier).
- ○ Determine the pattern of the response.
- ○ If a pattern is not detected or if the pattern does not correspond to any of the existing patterns, make the neural network learn the new pattern.
- ○ If the desired pattern is detected then there is no need for doing further tuning of the scaling factor.

Step 6 Send the output of the neural network (NN1) to the fuzzy scaling mechanism. Determine the increment changes in the scaling factor.
Step 7 Update the scaling factors. If the fine tuning mechanism is turned on, do the fine tuning before updating. Initially the scaling factors are set to a priori values.
Step 8 If the stability of the new rule base has to be checked, then apply it to NN2 (the system identifier). If the model's response is the desired one, then the rules are safe for applying it to the object system.
Step 9 Go to step 2 and perform the tuning cycle once again.

2.3. Real-time control of a direct drive motor

In order to perform real-time control, it is necessary that the controller stand alone with the sole task of calculating the output needed to control the object system. This means the task of communicating data for storing as well as acquiring controller parameters (if the controller is adaptive) should be performed by external processors. This way a real-time control can be achieved with required sampling rate for high bandwidth of operation. Typically, in order to achieve a particular closed loop bandwidth the sampling rate of the input parameters and control law updates has to be ten or more times faster than the bandwidth itself.

The fuzzy logic control algorithm requires processing of several operations such as fuzzification, inferencing and defuzzification. This means the computation time taken by the fuzzy controller itself does not leave any room for an adaptive algorithm such as rule generation, calculating the scale factor of the membership function, or neural network algorithms. In order to implement all these functionalities, a multi-processing architecture is needed. This can be achieved by combining a fast processor specifically meant for real-time processing, such as a TMS320C30 digital signal processor combined with a PC Intel processor (Pentium or 486). Fig. 3 shows the hardware built to interface and control a direct drive motor. The dynamics of a direct drive a.c. motor exhibits nonlinear characteristics. This is evident in the dead-zone regions of operation and frequency load characteristics. Industrial robots such as AdeptTwo use direct drive a.c. motors. The speed of the a.c. direct drive motor is a function of the frequency of the pulse signal on its field. This means that by varying the frequency of the a.c. power signal, the speed of the motor can be altered in an open loop configuration. This is achieved by a *voltage to frequency converter* which takes a d.c voltage from $+10$ V to -10 V to move the motor in either direction. A *digital to analog converter* of a resolution of 12 bits is used to interface to the DSP board through a memory mapped register I/O port via a tri-state buffer. The position of the motor is measured by an optical encoder (which generates 8000 pulses, or counts, per revolution of the motor). This is converted to digital 12-bit position data by encoder circuitry. A velocity signal is available which is digitized

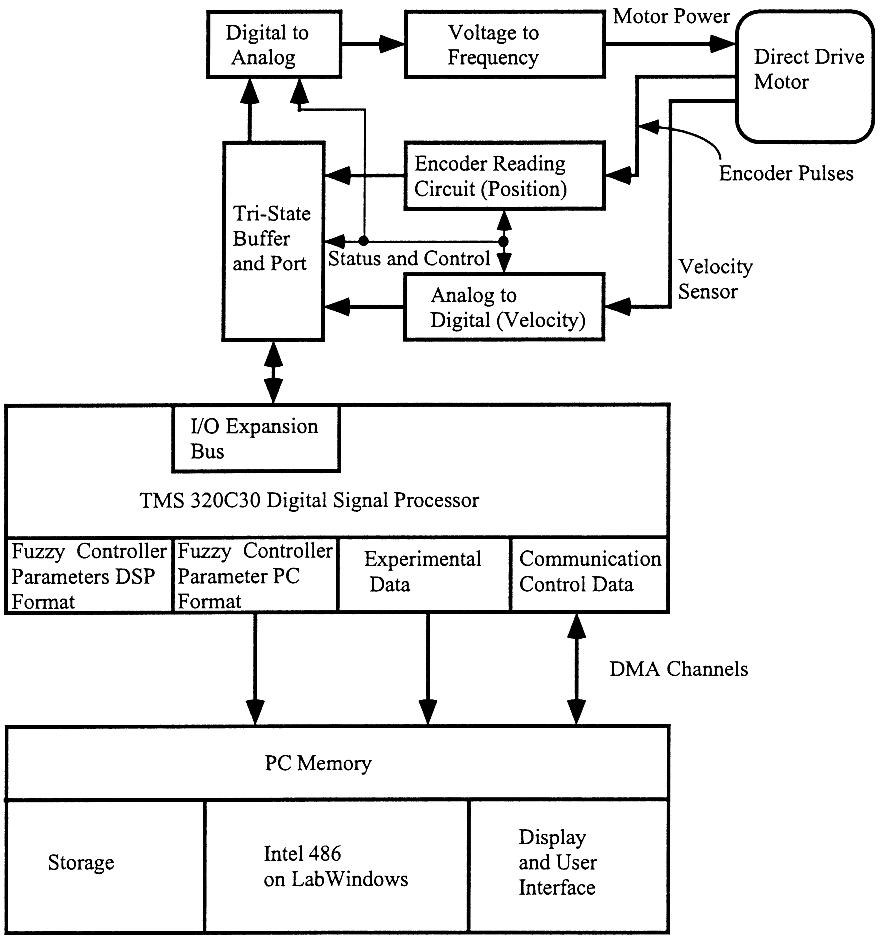

Digital to Analog
Voltage to Frequency
Motor Power
Direct Drive Motor
Encoder Reading Circuit (Position)
Encoder Pulses
Tri-State Buffer and Port
Status and Control
Analog to Digital (Velocity)
Velocity Sensor
I/O Expansion Bus
TMS 320C30 Digital Signal Processor
Fuzzy Controller Parameters DSP Format
Fuzzy Controller Parameter PC Format
Experimental Data
Communication Control Data
DMA Channels
PC Memory
Storage
Intel 486 on LabWindows
Display and User Interface

Fig. 3. Hardware for implementing neuro-fuzzy controller for real-time control of a direct drive motor using a digital signal processor.

using an *analog to digital converter* and interfaced to the DSP's I/O port. The DSP's expansion memory is accessible to the PC with which the data communication is carried out using *direct memory access* (DMA). The experimental data as well as fuzzy controller parameter and communication control data are sent back and forth using this DMA. The PC is interfaced with the user through a user friendly *Lab Windows* software interface.

The experiment of tuning the direct drive motor is carried out by generating new rules. The program running on the PC generates new fuzzy control parameters and updates the real-time fuzzy control algorithm running on the DSP. Based on the parameters of the new fuzzy controller the direct drive motor response is recorded. The last response is used to evaluate the new rules. These are again updated on the DSP memory and a new response is recorded. This tuning continues until the end of the tuning cycles. Here we describe only the effects of the rule generation mechanism. The effects of the fuzzy scaling mechanism are discussed in [10]. In order to perturb the direct drive motor we give a desired step change in the position of $+1000$ to -1000 encoder readings. For the 8000 count per revolution encoder, $+1000$ counts means $45°$ revolution of the motor, i.e. we are changing the desired position from $+45$ to $-45°$. Fig. 4 shows the result of the experiment. Each input variable has five symmetrical triangular-shaped membership functions. The rules can be interpreted as given by Eq. (1). Initially, there are no rules in the fuzzy controller. Hence for the first two cycles when the motor was commanded to go from $+1000$ to -1000 encoder readings, there is no action and the motor is stationary (0–1000 sampling time; each cycle corresponds to 500 sampled data). However, in the third cycle when the motor is commanded to go to $+1000$ counts, the motor spins out of control clockwise. This is because the rule generation mechanism has produced 4 new rules in the last two cycles and they are in action. These initial rules cause system instability and are not sufficient to control the motor adequately. This unstable behavior continues until after the 8th tuning cycle (after 4000 sampling instants). The corresponding motor command shows a bounded region. Fig. 5 shows the stabilized response. Here, the fuzzy controller has completely learned to control the direct drive motor.

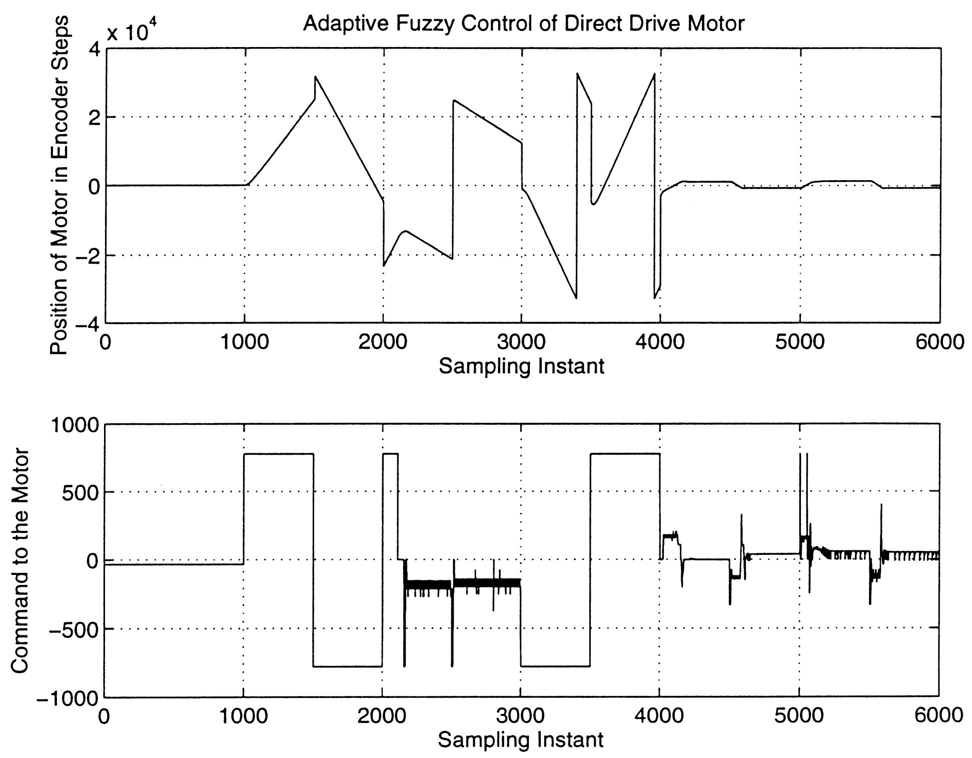


Fig. 4. Position control direct drive motor: response after first trial.

3. Genetic algorithms and fuzzy logic control

Genetic algorithms are robust optimization routines modeled after the mechanics of Darwinian theory of natural evolution [5]. GAs do not require gradient evaluation, hence they are applicable to solving a great range of optimization problems including determination of optimal parameters of a fuzzy logic rule-set. GAs have demonstrated the coding ability to represent parameters of fuzzy knowledge domains such as fuzzy rule sets and membership functions [11] in a genetic structure, and hence are applicable to optimization of fuzzy rule-sets. In this section, several issues pertaining to such integration of the two paradigms are discussed and illustrated through an application on real time hierarchical fuzzy control of a single-link flexible robotic arm.

To understand the actual mechanism of GAs, one may begin with its three most commonly used operators, namely: reproduction, crossover, and mutation. A member of a given population which has a higher fitness is given a higher chance to reproduce identical replicas of itself in an intermediate population. In this fashion, the optimization routine facilitates reproduction of higher fit individuals and hampers the reproduction of lower fit individuals. After reproduction, crossover randomly *mates* two individuals from an intermediate population and creates offsprings which are made up of a random combination of their parent's genetic code. For each generation, the process of crossover is repeated for all individuals in the

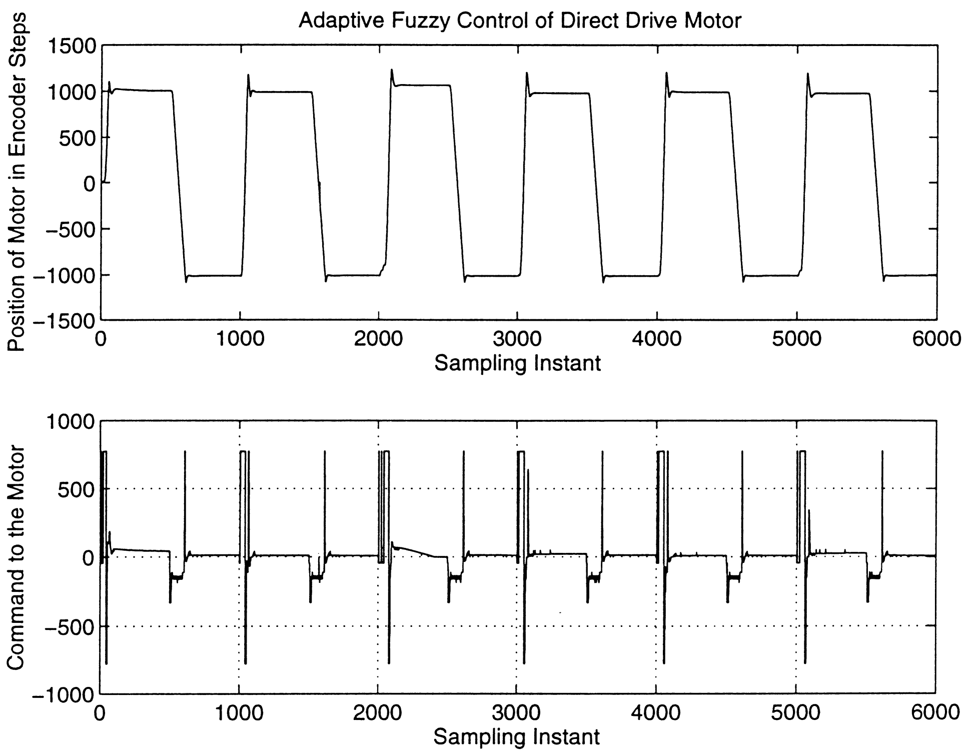


Fig. 5. Position control direct drive motor: response after final trial.

population. The population size is often a constant equal to the number of individuals in the initial population. The operations of reproduction and crossover create an environment where every generation benefits from the best genetic codes of the previous generations. However, if the building blocks for the optimal genetic structure is not in the initial population, these two genetic operators will be unable to find it. The last genetic operator, mutation, randomly mutates one or more of the values in the individual's genetic code in order to create diversity. The mutation operator allows for exploring *new* structures (directions of search) hence allowing the genetic optimization routine to invent new solution and finally locate the optimal solution even though the individuals in the initial population may not have contained the building blocks for the optimal solution.

When applying GA to optimization of fuzzy rule-sets, several questions arise. First is the design of the transformation function (the interpretation function) between the fuzzy knowledge domain (phenotype) and the GA coded domain (genotype). This is perhaps the most crucial stage of GA design and can significantly degrade the algorithm's performance if a poor or redundant set of parameters are chosen for a given optimization problem. Two important general categories of fuzzy expert knowledge consists of domain knowledge and meta-knowledge. Meta knowledge is the knowledge used in evaluating rules such as fuzzification (Scaling or λ cut), rule evaluation (such as Min/Max) and defuzzification (such as Max Membership, Centroid, or Weighted Average) methods. Relatively few research has been performed to study the effect of optimizing the meta-knowledge [12]. Most of the current research, as in this work, concentrate on optimizing parameters of the domain knowledge. The domain knowledge consists of the following two categories,

- Membership Function: General Shape (Triangular, Trapezoidal, Sigmoidal, Gaussian, etc.), Defining points (Center, Max Right, Min Left, etc.)
- Rule-Base: Fuzzy Associative Memory, Disjunctive (OR) and Conjunctive (AND) operations among antecedents in the rule base.

Even though various methods exist to encode both rule-base and membership functions in one GA representation, such coding can have several potential difficulties. In such situation, in addition to the level of complexity and large number of optimization parameter, the problem of competing conventions may arise and the landscape may unnecessarily become multi-modal. This is an important problem since there are often several (or many) fuzzy rule-sets which can represent a given nonlinear function. This means that there are more than one optimal solution to a given optimization problem which raises the issue of multi-modality for fuzzy logic systems, or more specifically competing conventions where different chromosomes in the representation space have the same interpretation in the evaluation space. When designing the interpretation function, therefore, the coding needs to contain fewest possible parameters to avoid the problem of dual representation, and yet the coding needs to have enough complexity to contain all possible optimal or near optimal solutions. Evolutionary approaches such as Niched GA [13] are designed to search in complex multi-modal landscapes.

As a result, in the present approach, this problem is attended to by limiting the optimization parameter space to membership function parameters only. This was a design decision which was made considering the following two considerations.

- The problem of multi-modality is introduced when the GA string contains both parameters

of membership functions and rules.
- In control of most physical systems, rules can often be derived either intuitively or through operator experience. The ambiguous and *fuzzy* portion of a knowledge base is often the membership functions.

For simplicity in coding the simulation and the real-time algorithms, only triangular membership functions are coded for optimization here. Fig. 6 illustrates a triangular membership function whose three determining parameters (a,b,c) are shown. Assuming a normalized membership function, the three parameters are real numbers between -1 and 1. The coding in GA is performed as follows, the real parameter a is first mapped to an n-bit signed binary string where the highest bit represents the sign, see Fig. 7. This way, the parameter a can take on 2^n different values. Then the binary number is aggregated with other n-bit binary numbers to construct the phenotype representation.

The second issue which arises is how to utilize initial expert knowledge for a better and faster convergence. In other search routines such as hill-climbing, it is clear that starting from a "good" point can significantly improve computation time needed for convergence to an optimal solution. However, the conventional GA applications generate a random initial population without using any a priori expert knowledge. This, in general, will provide a more diverse population while sacrificing convergence time. This convention can indeed be adequate if there is no a priori knowledge as to where a "good" solution may exist. However, in fuzzy logic applications, there is usually access to some expert knowledge which, even though it may not be the optimal solution, is often a reasonably good solution. Schultz [14] addressed the problem of incorporating a priori knowledge by introducing two types of populations, homogeneous and heterogeneous. The homogeneous population consists of individuals created randomly while their string is augmented with the same priori rules. In this sense, they are all identical and hence homogeneous. He concluded that a trade-off exists between manual knowledge and machine learning. The heterogeneous population consists of members which are not identical. This is also referred to as the "seeding" technique. In Ref. [15] the process of *seeding* the initial population with one or more experts' knowledge is proposed. The few seeded chromosomes have the chance of reproducing through mutation and crossover with other randomly generated chromosomes in the population. This method improves the performance of GA by providing the genetic population with a set of highly fit building blocks, as compared with GAs starting with random initial populations. However, such population still

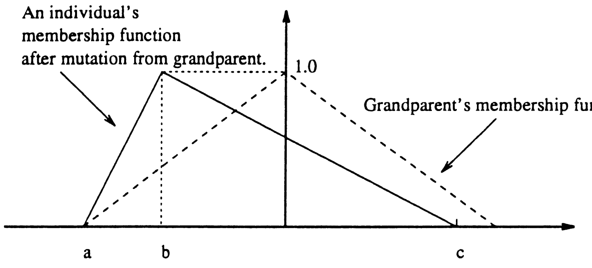

An individual's membership function after mutation from grandparent.
1.0
Grandparent's membership function
a
b
c

Fig. 6. Triangular membership function with three parameters a,b,c.

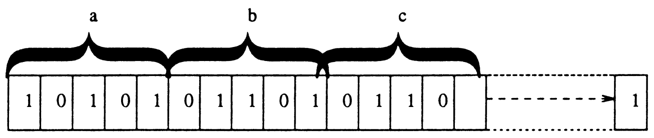


Fig. 7. Chromosome coding in representation space.

requires a large number of iterations before convergence since the "useful" schemata exist in only one or few seeded members and can only be reproduced as fast as the rate of reproduction. Akbarzadeh [16] proposed the *grand-parenting* scheme where the initial population is comprised of mutations of the "knowledgeable" grandparent. This scheme takes advantage of expert knowledge while maintaining diversity necessary for an effective search. Through grandparenting, an expert's a priori knowledge can be utilized to improve fitness of GA's initial population, thereby increasing the speed and performance of the search routine. In the present approach, the method of grand-parenting is used to improve the convergence rate of the GA optimization process.

The third issue is defining a fitness function. A fitness function is a very important aspect of GA design since it determines the direction of the search. Fitness functions come in as many different forms as the systems which they are optimizing. In general, for a lumped parameter system (such as a flexible robot arm), parameters such as control effort $u(t)$, rise-time t_r, overshoot γ, and steady-state error e_{ss} are usually incorporated in a quadratic fitness function. Often, constant multipliers define the relative degree of importance which is given to a certain parameter compared to others. The following equation is a typical equation which gives a higher fitness to improved responses with lower overshoot, control effort, and steady state error.

$$f_{\text{fitness}} = \int_{t_i}^{t_f} \frac{1}{e_{ss}^2 + \gamma^2 + u(t)^2 + 1} \mathrm{d}t. \tag{5}$$

3.1. Application to flexible robot control

The above concepts are now applied to controller optimization of a flexible link robotic system as shown in Fig. 8. The flexible link is a distributed parameter system with spatial as well as temporal parameters. In other words, the states of a flexible robotic system are functions of both space and time. This complicates the modeling of the system and, consequently, the process of designing the controller. Due to the complexity of a mathematical representation for such systems, fuzzy logic is considered an attractive alternative to their control. One of the issues in development of fuzzy controllers is determining faithful expert knowledge. Expert knowledge, however, is difficult to produce since there is often no human expert to consult and training a human expert may not be a feasible alternative due to cost and other practical considerations. Furthermore, human psychological issues may prohibit a faithful reproduction of a rule-base from an expert. In addition, the unstructured operating environments associated with space and waste handling projects require the robot controller to also adapt to changing conditions. In the process of designing fuzzy rule sets, membership

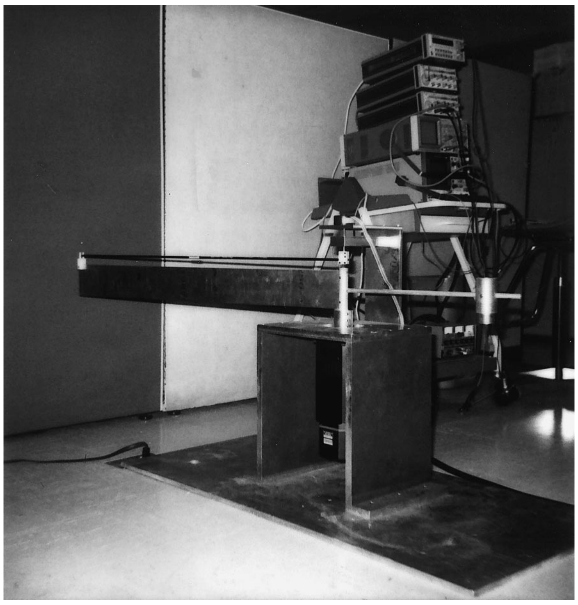


Fig. 8. A single link flexible robot arm.

functions are often chosen through an ad hoc process of random selection and evaluation. As a viable alternative, good results have been achieved by employing genetic algorithms to tune membership parameters within a fuzzy controller's knowledge base [17]. Genetic algorithms equip the fuzzy controller with some evolutionary means by which it can improve its rule-base when faced with inadequate a priori expert knowledge or varying circumstances in its operating environment.

The GA-learning hierarchical fuzzy control architecture is shown in Fig. 9. Within the hierarchical control architecture, the higher level module serves as a fuzzy classifier by determining spatial features of the arm such as *straight, oscillatory, curved.* This information is supplied to the lower level of hierarchy where it is processed among other sensory information such as errors in position and velocity for the purpose of determining a desirable control input (torque). In [18] this control system is simulated using only a priori expert knowledge.

To demonstrate the usage of genetic algorithms, GA is applied to optimize parameters related to input membership functions of the higher level of hierarchy. Other parameters in the knowledge base are not allowed to vary. The following fitness function was used to evaluate various individuals within a population of potential solutions,

$$\text{fitness} = \int_{t_i}^{t_f} \frac{1}{e^2 + \gamma^2 + 1} \mathrm{d}t, \tag{6}$$

where e represents the error in angular position and γ represents overshoot. Consequently, a fitter individual is an individual with a lower overshoot and a lower overall error (shorter rise time) in its time response.

Here, results from previous simulations of the architecture are applied experimentally. The

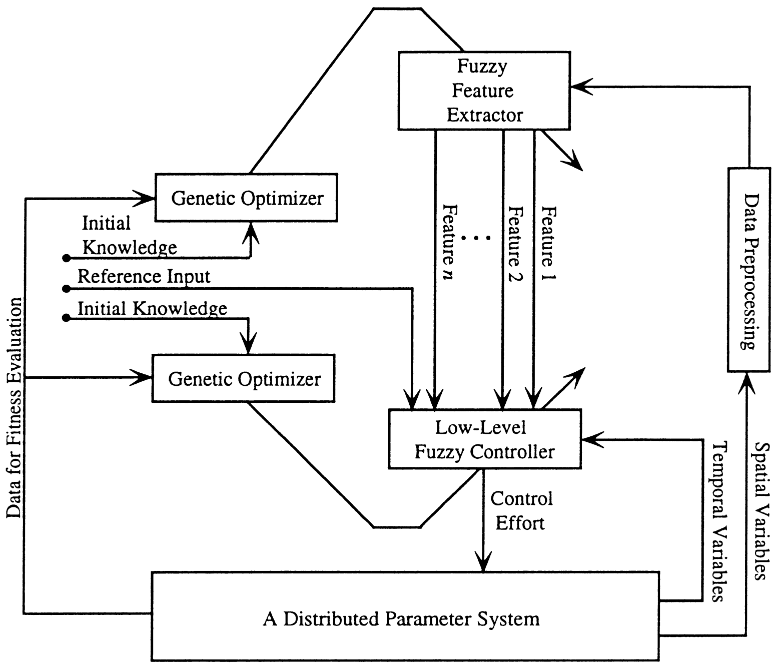


Fig. 9. GA-based learning hierarchical control architecture.

method of *grand-parenting* [16] was used to create the initial population. Members of the initial population are created through mutation of the knowledgeable *grandparent(s)*. As a result, a higher fit initial population results in a faster rate of convergence as is exhibited in Fig. 10(a). Fig. 10(a) shows the time response of the GA-optimized controller when compared to previously obtained results through the non-GA fuzzy controller. The rise time is improved by 0.34 s (an 11% improvement), and the overshoot is reduced by 0.07 radians (a 54% improvement). The average fitness of each generation is shown in Fig. 10(b). A total of 10 generations were simulated. The mutation rate for creating the initial population was set at 0.1.

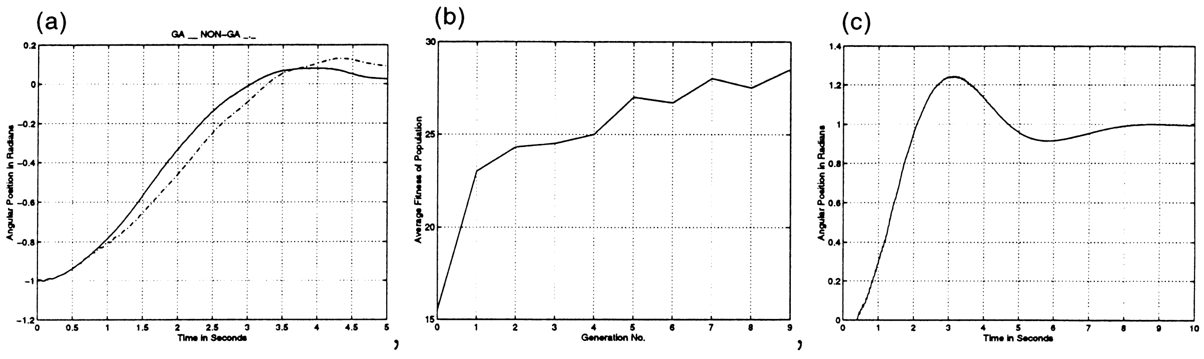


Fig. 10. GA simulation. (a) comparison of simulation responses; (b) plot of average fitness; and (c) initial experimental results.

This value was chosen to increase diversity among members of initial population. GA depends on this diversity to exploit a large number of differing path of solutions in parallel. The mutation rate throughout the rest of the simulation, however, was set to 0.01. Since a high mutation rate delays convergence. The probability of crossover was set to 0.6. Initial experimental results demonstrate that the GA-learned controller is able to control the actual experimental system as in Fig. 10(c).

The hardware used to implement the above algorithms is the same as was explained in Section 2 with a few modifications pertaining to flexible robot control such as a tip end position sensor and several strain gauges distributed evenly across the length of the flexible beam. Control update was performed at 250 Hz.

4. Genetic programming of fuzzy navigation behavior

In this section, we describe the application of GP to the problem of learning/discovering rules for use in a fuzzy rule-based behavior control system. Like GA, GP computationally simulates the Darwinian evolution process by applying fitness-based selection and genetic operators to a population of individuals. However, in this case each individual represents a computer program of a given programming language, and is a candidate solution to a particular problem. The programs are structured as hierarchical compositions of functions (in a set F) and terminals (function arguments in a set T). These individuals participate in a probabilistic evolutionary process wherein the population evolves over time in response to selective pressure induced by the relative fitness of the individuals for solving the problem.

For the purpose of evolving fuzzy rule-bases, the search space is contained in the set of all possible rule-bases that can be composed recursively from F and T. The set, F, consists of components of the generic *if-then* rule and common fuzzy logic connectives, i.e. functions for antecedents, consequents, fuzzy intersection, rule inference, and fuzzy union [19]. The set, T, is made up of the input and output linguistic variables and the corresponding membership functions associated with the problem. A rule-base that could potentially evolve from F and T can be expressed as a tree data structure with symbolic elements of F occupying internal nodes, and symbolic elements of T as leaf nodes of the tree. This tree structure of symbolic elements is the main feature which distinguishes GP from GAs which use the numerical string representation. All rule-bases in the initial population are randomly created, but descendant populations are created primarily by reproduction and crossover operations on rule-base tree structures.

GP cycles through the current population evaluating the fitness of each rule-base based on its performance in computer simulations of the fuzzy control system. After a numerical fitness is determined for each rule-base, genetic operators are applied to the fittest rule-bases to create a new population. The cycle repeats on a generation by generation basis until satisfaction of termination criteria (e.g. lack of improvement, maximum generation reached, etc). The GP result is the best-fit rule-base that appeared in any generation.

The symbolic data processing done by GP makes it particularly amenable to automatic evolution of fuzzy rules, which are comprised of symbols representing fuzzy sets and fuzzy logic connectives. In the GP approach to evolution of fuzzy rule-bases, the same fuzzy linguistic terms and operators that comprise the genes and chromosome of the rule-base persist in the phenotype. Thus, the use of GP allows direct manipulation of the actual linguistic rule representation of fuzzy rule-based systems. Furthermore, the dynamic variability of the tree representation permits populations with rule-bases of various sizes and different numbers of rules. This enhances population diversity which is important for the success of a GP system, and any evolutionary algorithm for that matter. We have applied GP to evolve fuzzy rule-bases that are used to coordinate multiple fuzzy-behaviors arranged in a hierarchical structure. This hierarchy of fuzzy-behaviors has been employed for autonomous control of mobile robot navigation. Before presenting application results, we describe the behavior hierarchy to clarify the context in which GP was used and our motivation for doing so.

4.1. Fuzzy-behavior hierarchy

Under the behavior control paradigm, one decomposes high-level robot motion control into a set of special-purpose behaviors that achieve distinct tasks when subject to particular stimuli. Fuzzy-behaviors are synthesized as fuzzy logic rule-bases, i.e. collections of a finite set of fuzzy if-then rules. Each behavior is encoded as a fuzzy rule-base with a distinct mobile robot control policy governed by fuzzy inference. Thus, each fuzzy-behavior is similar to a conventional fuzzy logic controller in that it performs an inference mapping from some input space to some output space. If X and Y are input and output universes of discourse of a behavior with a rule-base of size n, the usual fuzzy if-then rule takes the following form

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } y \text{ is } \tilde{B}_i \tag{7}$$

where x and y represent input and output fuzzy linguistic variables, respectively, and $\tilde{A}_i$ and $\tilde{B}_i$ ($i = 1 \dots n$) are fuzzy subsets representing linguistic values of x and y. In the mobile robot controller, the input x refers to sensory data or goal information; y refers to motor control signals or behavior activation levels. In general, the rule antecedent consisting of the proposition "x is $\tilde{A}_i$" could be replaced by a conjunction of similar propositions; the same holds for the rule consequent "y is $\tilde{B}_i$". Fuzzy-behaviors consist of a finite set of such rules.

In our fuzzy-behavior hierarchy, a collection of *primitive behaviors* resides at the lowest level. These are simple, self-contained fuzzy controllers that serve a single purpose by operating in a reactive or reflexive fashion. Via fuzzy infer ence, they perform nonlinear mappings from different subsets of the robot's sensor suite to (typically, but not necessarily) common actuators. As designed, a single behavior operating alone would be insufficient for autonomous navigation tasks. Such primitive behaviors are building blocks for more intelligent *composite behaviors* implemented as fuzzy decision systems. That is, they are combined synergistically to produce intelligent behavior(s) suitable for accomplishing goal-directed navigation.

Indoor mobile robots should be capable of collision-free navigation in static and dynamic environments. Several capabilities are necessary to achieve this including collision avoidance, self and goal localization, and traversal through indoor features such as halls, doorways, and densely cluttered spaces. A behavior hierarchy encompassing these capabilities is shown in Fig. 11. This figure implies that goal-directed navigation can be decomposed as a behavioral function of `goal-seek` (collision-free navigation to some location) and `route-follow`. These behaviors can be further decomposed into the primitive behaviors shown, with dependencies indicated by the adjoining lines. The purposes of `wall-follow` and `avoid-collision` are implied by their names. The `doorway` behavior implies one that can guide a robot through narrow passageways in walls. The `go-to-xy` behavior will direct a robot to navigate along a straight line trajectory to a particular location.

In Fig. 11, interconnecting circles between composite behaviors and the primitive level represent the weights and activation thresholds used to concurrently coordinate of the associated primitive behaviors. Behavior coordination is achieved by a mechanism of weighted control decision-making embodied in a concept called the *degree of applicability* (DOA) — a measure of the instantaneous level of activation of a motion behavior. Fuzzy rules of composite behaviors are formulated to include weighting consequents which modulate the DOAs of primitive behaviors at a lower level. We refer to these as applicability rules. The DOA, α, of a primitive behavior is specified in the consequent of applicability rules of the form

$$\text{IF } x \text{ is } \tilde{A}_i \text{ THEN } \alpha \text{ is } \tilde{D}_i \tag{8}$$

where $\tilde{A}_i$ is defined as in (7). $\tilde{D}_i$ is a fuzzy

subset representing the linguistic value (e.g. *high*, low, etc) of the behavior's DOA to the situation prevailing during the current control cycle. It is defined over the closed unit interval [0, 1]. This feature allows certain robot behaviors to influence the overall behavior to a greater or lesser degree as required by the current situation and goal. Since control recommendations from each applicable behavior are considered in the final decision, the resultant control action can be thought of as a consensus of recommendations offered by multiple experts. This strategy operates as a form of adaptation since it causes the control policy to dynamically change in response to goals, sensory input, and internal state. The behavior hierarchy, then, is a dynamic nonlinear mapping from situations to actions rather than a static nonlinear mapping represented by a fixed set of fuzzy rules.

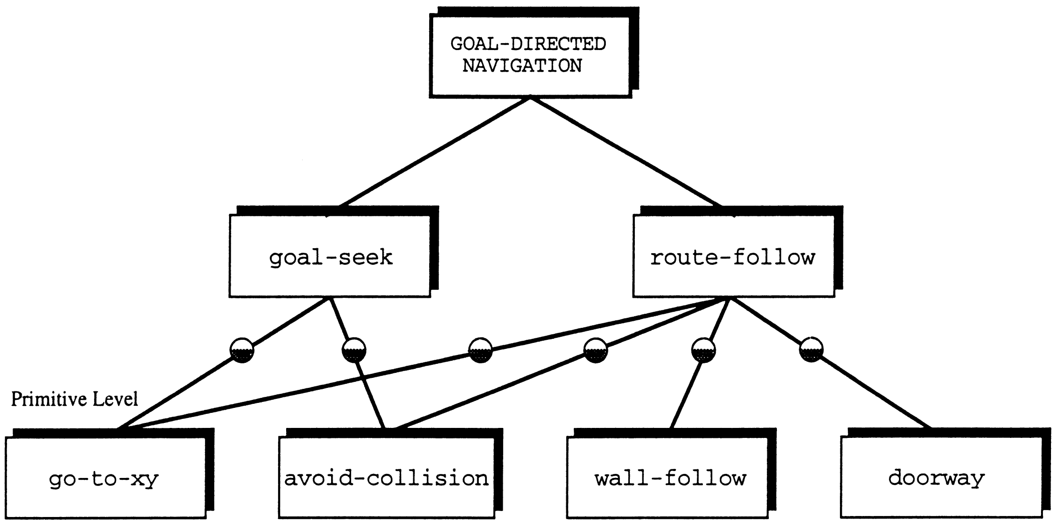


Fig. 11. Mobile robot behavior hierarchy.

4.1.1. Related work

During execution of adaptive behavior, complex interactions occur when more than one primitive behavior is active. When behaviors compete for control of the robot by recommending different control actions for common actuators, the problem becomes one of coordinating multiple task-achieving behaviors and resolving conflicts amongst them. In order to formulate suitable coordination rules to implement the strategy described above, one must first decide what the DOAs of low-level behaviors should be in all practical situations perceived from sensory input. Formulation of such rules for coordination is not entirely intuitive, and expert knowledge about how to concurrently coordinate primitive behaviors is not readily available. In fact, in the absence of experts or sufficient knowledge of the problem, the design of knowledge-based control systems with interacting rule-bases is often met with difficulty. In such cases, a means of automatic discovery/learning of coordination rules has great utility for developing robust knowledge-based controllers. As such, it is desirable to supplement the behavior coordination strategy described above with a technique for learning its governing rules.

Several instances of independent research have converged to similar ways of approaching multiple behavior coordination [20–23]. However, these efforts have not proposed approaches to learning coordination rules. Elsewhere, the rule learning problem has been approached in the contexts of other coordination schemes by using reinforcement learning [24] and hybrids of reinforcement and neural networks [25,26]. In [19], we successfully applied GP for learning fuzzy rule-bases for low-level regulation and tracking types of problems. Using this earlier success as a foundation, we apply our GP approach here to learn higher-level behavior coordination rules. In particular, a fuzzy coordination behavior for goal seeking, comprised of rules in the form of Eq. (8),is what we wish to evolve.

4.2. Rule-base evolution for behavior coordination

Thus far we have described the ingredients of a computing system that incorporates fuzzy logic control, with rule learning by GP, into the framework of a behavior hierarchy. A block

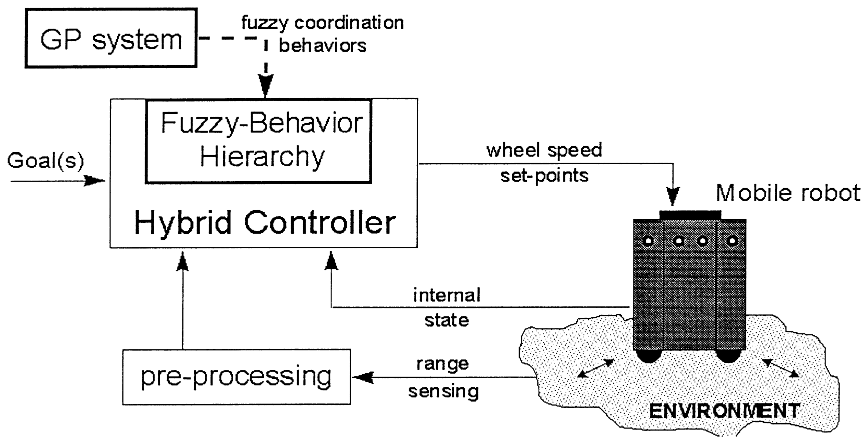


Fig. 12. GP-fuzzy mobile robot control architecture.

diagram illustrating the general data flow in this GP-fuzzy control architecture is shown in Fig. 12. The arrow, in the figure, delivering GP-evolved coordination behaviors to the behavior hierarchy is dashed to indicate that these behaviors are learned/evolved off-line. The laboratory testbed, called LOBOt, is a custom-built mobile robot driven by a two-wheel differential configuration with two passive casters for support. The independent drive motors are geared d.c. motors. As shown in Fig. 13, LOBOt is octagonal in shape; it stands about 75 cm tall and measures about 60 cm in width. The outputs from the behavior hierarchy are right and left wheel speeds; the inputs to the hierarchy are the goal location and subsets of sensor readings. Range sensing is achieved using a layout of 16 ultrasonic transducers (mounted primarily on the front and sides); optical encoders on each driven wheel provide position information used for dead-reckoning. Its control computing environment includes a 75 MHz Pentium-based master processor (laptop PC) and Motorola MC68HC11 microprocessor slaves for sonar processing and low-level motor control functions.

GP was used to evolve the necessary coordination rules for LOBOt based on the approach described above. In the current implementation, applicability rules used by `goal-seek` to coordinate the underlying primitive behaviors consider three instantaneous input states — the range to the nearest obstacle, the distance from the goal, and the angular heading to the goal. The consequents of these rules prescribe a DOA for each relevant primitive behavior.

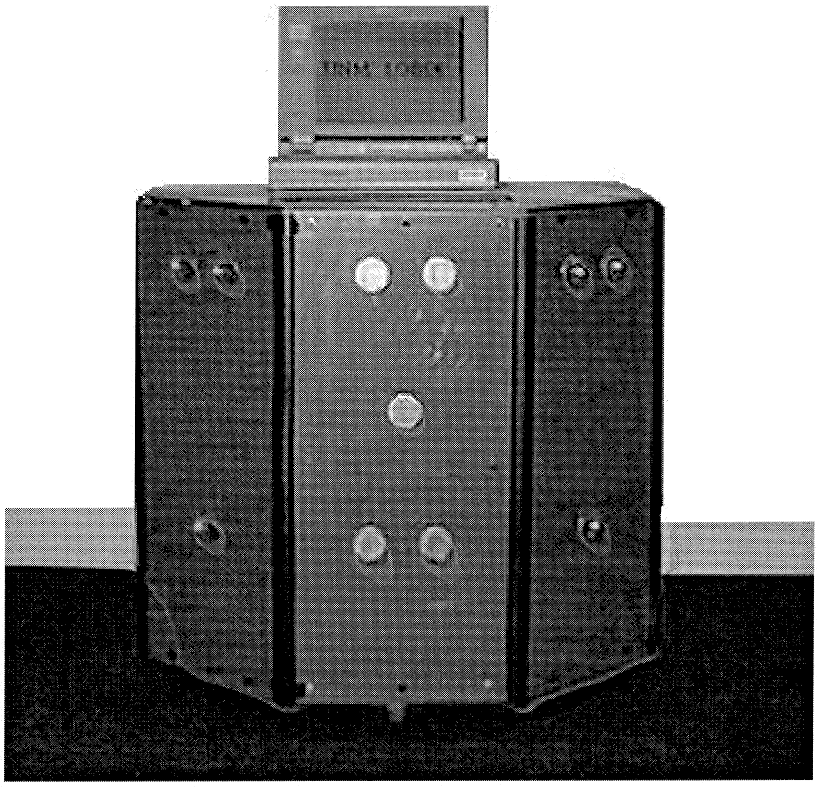


Fig. 13. UNM LOBOt.

4.2.1. Behavior fitness evaluation

During off-line evolution, each behavior in the current population is evaluated via simulation in a number of indoor fitness cases subject to an upper time limit of 200 s. In this work, $n_f = 5$ fitness cases are used; the simplest and most difficult of these are illustrated in Fig. 14(a) and (b), respectively. Goal locations in the figure are indicated by an $\times$, the robot is depicted as an octagonal icon with a radial line designating its initial heading, and its range sensor horizon is indicated by the shaded regions of Fig. 14(a).

In each case, the dimension of the indoor space is 10 m $\times$ 10 m. Each fitness case was chosen to represent situations likely to be encountered in indoor environments. For a given behavior, the score of a trial run through fitness case i is given by

$$S_i = \begin{cases} 100 & ; \text{ goal reached} \\ \dfrac{100}{\gamma(1 + 10e_{\mathrm{N}})} & ; \text{ otherwise} \end{cases} \tag{9}$$

where e_{N} is the normalized residual distance to the goal in the case of a time-out or collision. The parameter $\gamma = 2$ if a collision occurs; otherwise $\gamma = 1$. That is, the score for an unsafe trial is half of that for a collision-free trial with all else being equal (see Fig. 15). The overall fitness of the behavior is the average score over all n_f fitness cases:

$$F = \frac{1}{n_f}\sum_{i=1}^{n_f} S_i. \tag{10}$$

Thus, the highest possible score, and hence fitness, is 100. In evolutionary algorithms, such as GP, it is important that the fitness function map observable parameters of the problem into a spectrum of values that differentiate the performance of individuals in the population. If the spectrum of fitness values is not sufficiently rich, the fitness function may not provide enough information to guide GP toward regions of the search space where improved solutions might be found. The fitness function and score were formulated with this in mind, and also to reward behaviors responsible for consistently reaching, or coming within close proximity to, the goals.

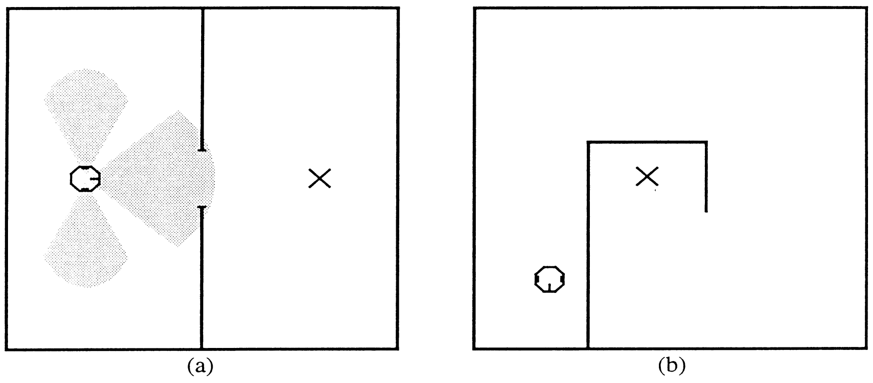


Fig. 14. Example fitness cases.

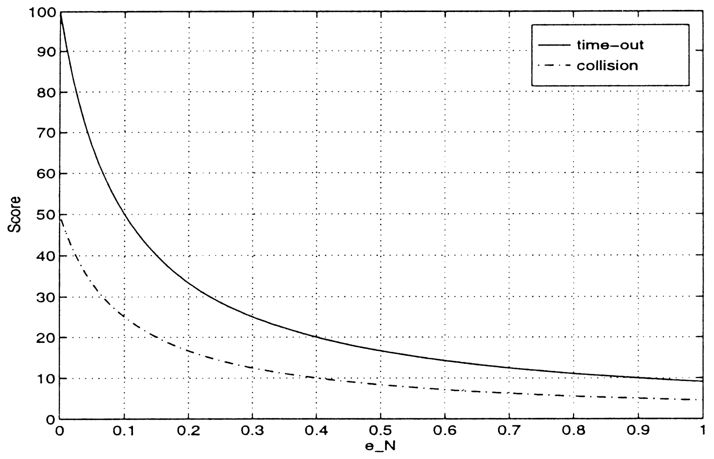


Fig. 15. Behavior fitness case scoring function.

4.2.2. Behavior evolution

The best behaviors evolved off-line by GP were tested in a simulated indoor environment that is considerably more general than any one of the fitness cases used during the evolution process. Such an environment is suitable for testing the generalization capability of the evolved behaviors. As we will see shortly, the test environment also differs from the physical environment in which LOBOt actually operates.

Population sizes of 10–20 rule-bases were run for a number of generations ranging from 10 to 15. In GP, genetic diversity remains high even for very small populations due to the tree structure of individuals [6]. The mean performance of GP over five runs is shown, in the left half of Fig. 16, as the progression of the population average fitness during the first ten generations. A trend towards higher fitness is evident. Evolved coordination behaviors were

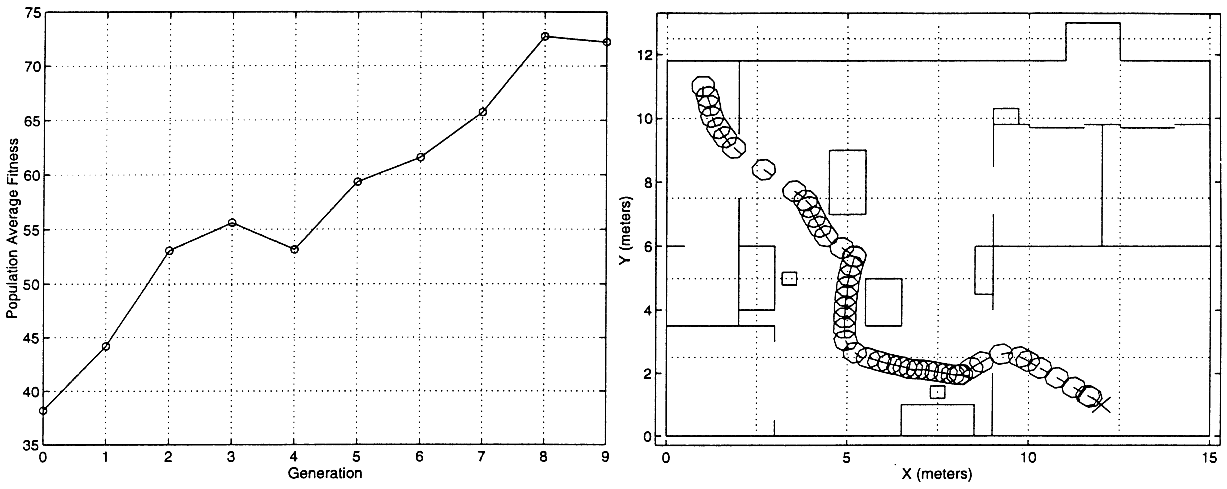


Fig. 16. Performance of GP-evolved coordination.

tested in the simulated environment (Fig. 16) for sensor-based navigation from initial pose (1 11 $-\pi/2$) to a goal located at (12,1). The right half of the figure shows a navigation run successfully coordinated by a GP-evolved behavior comprised of 12 fuzzy rules that specify DOAs for the underlying primitive motion behaviors (see Fig. 11).

4.2.3. Real-time application

Once a suitable coordination behavior is learned it can be ported to the actual robot to support navigation control in its physical operating environment. For this application, the cycle time of the control system was 0.15 s (7 Hz). This time includes the time spent acquiring and preprocessing sonar and encoder data, and commanding the motors. The maximum speed of the mobile robot was limited to 0.3 m/s.

During operation, the robot is not provided with an explicit map of the environment. However, it is cognizant of the notion of a two-dimensional Cartesian coordinate system. Its paths are not pre-planned; they are executed in response to instantaneous sensory feedback from the environment. Therefore, we are essentially dealing with a local navigation problem as opposed to a global navigation problem which relies on a global map that is either provided a priori, or is acquired via exploration. A representative result of goal-seeking without prior map knowledge is presented here. The experiment was conducted in an indoor environment consisting of corridors and doors. The robot's task is to navigate from one location to another on the same floor of the building. The result of the navigation task is shown in Fig. 17 as the path traveled by LOBOt in a portion of the indoor environment which includes the start and commanded goal. The robot was commanded to navigate from a start pose $(x, y, \theta) = (9.5$ m, 22 m, 3.0 rad) to a goal located at $(x, y) = (21.5$ m, 37.5 m). As shown in Fig. 17, LOBOt successfully navigates within close proximity to the goal without prior map-based knowledge using only ultrasonic sensing and approximate dead-reckoning.

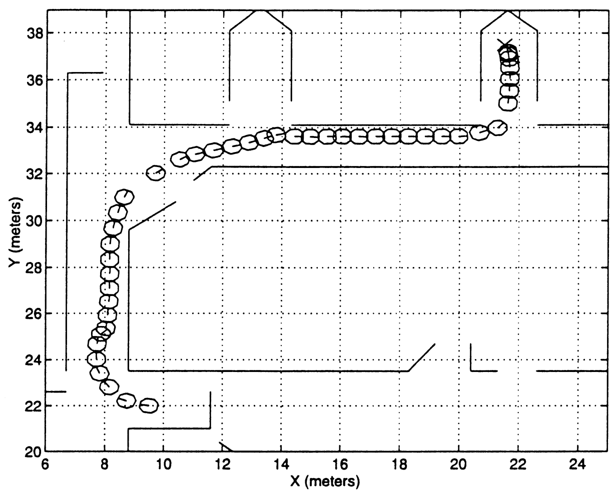


Fig. 17. Real-time goal-seeking task.

5. Conclusion

In this paper, three experiments illustrate the utility of soft-computing approaches in handling complex models and unstructured environments. Neuro-fuzzy [9], GA-fuzzy [27], and GP-fuzzy [28] hybrid paradigms are successfully implemented to solve three prominent robot control issues, namely: control of direct drive motors, control of flexible links, and intelligent navigation of mobile robots. In the future, as these paradigms mature, we will gain more knowledge of their exact nature and advantages. This will allow us to combine soft computing paradigms for more intelligent and robust control. Not long ago, a hybrid combination of these paradigms could not be applied to a real-time system. However, as shown in this paper, with the current advances in increase of speed of processing and DSP parallel processors, various combination of hybrid soft computing paradigms are now realizable.

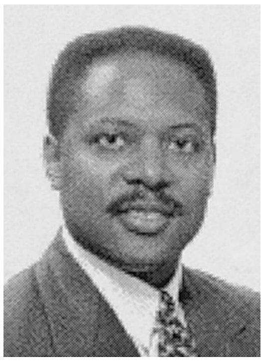
References

[1] Wang L-X. Adaptive fuzzy systems and control: design and stability analysis. Englewoods Cliffs, NJ: Prentice Hall, 1994.
[2] Jamshidi M. Large-scale systems: modeling, control and fuzzy logic. Englewoods Cliffs, NJ: Prentice Hall, 1997.
[3] Mamdani EH. Twenty years of fuzzy control: experiences gained and lessons learnt. In: IEEE Intl. Conf. on Fuzzy Systems, 1993. p. 339–44.
[4] Aminzadeh F, Jamshidi M. Soft computing: fuzzy logic, neural networks, and distributed artificial intelligence. Englewoods Cliffs, NJ: Prentice Hall, 1994.
[5] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley, 1989.
[6] Koza JR. Genetic programming: on the programming of computers by means of natural selection. Cambridge, MA: MIT Press, 1992.
[7] Texas Instruments. TMS320C3x User's Guide.
[8] DSP Research. TIGER 30 Reference Manual.
[9] Kumbla K. Adaptive neuro-fuzzy controller for passive nonlinear systems. Ph.D. Dissertation, University of New Mexico, 1997.
[10] Kumbla K, Akbarzadeh M, Medina E, Young K Adaptive neuro-fuzzy controller on a digital signal processor. vol. 5. World Automation Congress, Montpelier, France, 1996. p. 435–40.
[11] Homaifar A, McCormick E. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. IEEE Transactions on Fuzzy Systems 1995;3(2):129.
[12] Smith M. Dynamic tuning of parametrized defuzzification methods applied to automatic control and diagnosis. In: Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, New Orleans, LA, Sept. 8–11, 1996. p. 707–13.
[13] Chambers L. Practical handbook of genetic algorithms: new frontiers. Boca Raton FL: CRC Press, 1995.
[14] Schultz AC, Grefenstette JJ. Improving tactical plans with genetic algorithms. In: Proceedings of the 2nd International Conference on Tools for AI, Herndon, VA, 1990.
[15] Lee MA, Takagi H. Embedding apriori knowledge into an integrated fuzzy system design method based on genetic algorithms. In: Proceedings of the 5th IFSA World Congress, 1993.
[16] Akbarzadeh-T M-R, Jamshidi M. Incorporating a priori expert knowledge in genetic algorithms. In: Proceedings of the IEEE Conference on Computational Intelligence in Robotics and Automation, Monterey, CA, 1997.
[17] Lee MA, Takagi H. Integrating design stages of fuzzy systems using genetic algorithms. In: Proceedings of the 1993 IEEE International Conference on Fuzzy Systems, San Francisco, CA, 1993. p. 612–7.

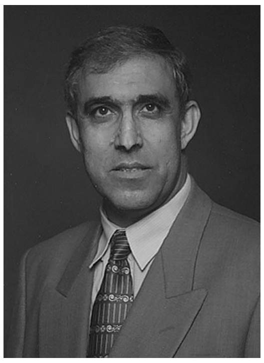
[18] Akbarzadeh-T M-R. A fuzzy hierarchical controller for a single flexible arm. In: Proceedings of the 1994 International Symposium on. ISRAM'94, Maui, Hawaii: Robotics and Manufacturing, 1994.
[19] Tunstel E, Jamshidi M. On genetic programming of fuzzy rule-based systems for intelligent control. Proceedings of the International Journal of Intelligent Automation & Soft Computing 1996;2(3):273–84.
[20] Saffiotti A, Konolige K, Ruspini EH. A multivalued logic approach to integrating planning and control. Artificial Intelligence 1993;12:481–526.
[21] Moreno L, Moraleda E, Salichs MA, Pimentel JR, de la Escalera A. Fuzzy supervisor for behavioral control of autonomous systems. In: Intl. Conf. on Industrial Electronics, Control, and Instrumentation, 1993. p. 258–61.
[22] Michaud F, Lachiver G, Le Dinh CT. Fuzzy selection and blending of behaviors for situated autonomous agent. In: IEEE Intl. Conf. on Fuzzy Systems, 1996. p. 258–64.
[23] Correia L, Steiger-Garção A. A useful autonomous vehicle with a hierarchical behavior control. In: Móran F, Moreno A, Merelo JJ, Chácon P, editors. Advances in Artificial Life, 3rd European Conf. on Artificial Life. Granada: Springer-Verlag, 1995. p. 625–39.
[24] Bonarini A. Learning to coordinate fuzzy behaviors for autonomous agents. In: 2nd European Congress on Intelligent Techniques and. Soft Computing EUFIT'94, 1994. p. 475–9.
[25] Beom HR, Koh KC, Cho HS. Behavioral control in mobile robot navigation using fuzzy decision making approach. In: IEEE Intl. Conf. on Robots and Systems, 1994. p. 1938–45.
[26] Gachet D, Salichs MA, Moreno L, Pimentel JR. Learning emergent tasks for an autonomous mobile robot. In: IEEE Intl. Conf. on Robots and Systems, 1994. p. 290–7.
[27] Akbarzadeh-T M-R. Fuzzy control and evolutionary optimization of complex systems. Ph.D. Dissertation, University of New Mexico, 1998.
[28] Tunstel E. Adaptive hierarchy of distributed fuzzy control: Application to behavior control of rovers. Ph.D. Dissertation, University of New Mexico, 1996.

Mohammad-R. Akbarzadeh-T received the B.S., M.S. and Ph.D. degrees in electrical engineering, with concentration in robotics and control systems, from University of New Mexico. His dissertation addressed the utility of fuzzy logic and genetic algorithms in control of complex systems. In 1996, he joined Center for Autonomous Control Engineering as a research engineer. In this capacity, he has since initiated a software team aimed at developing a user friendly software environment for soft computing, SoftLab. Currently, he holds dual appointments as a research engineer at Center for Autonomous Control Engineering as well as an adjunct assistant professor at the department of electrical and computer engineering, University of New Mexico. His research interests include robotics, control systems, cooperative robots, power systems, desalination processes, and other applications of soft computing paradigms to intelligent control of systems.

Kishan Kumar Kumbla graduated with a Bachelor of Engineering degree, with distinction from the University of Mysore, India in 1985. He joined the National Thermal Power Corporation in India and worked there as an instrumentation and control engineer. His pursued his graduate studies at the University of Tennessee with a Master in Electrical Engineering in 1992. He got his Ph.D. in Electrical Engineering from the University of New Mexico in 1997. His interests are in neural networks, fuzzy control, genetic algorithms and real-time control algorithms. He worked for air force research laboratory, Albuquerque implementing real-time control for micro-electro-mechanical (MEM) devices. Currently he is working for IBM in San Jose where he is with the advanced servo-writing group for hard disk drives.



Eddie Tunstel received the B.S. and M.E. degrees in mechanical engineering, with a concentration in robotics, from Howard University, Washington, DC. His thesis addressed the use of symbolic computation for automated modeling of robotic manipulators. In 1989 he joined the Robotic Intelligence Group at JPL supporting research and development activities on NASA Planetary Rover projects. As a JPL Fellow he received the Ph.D. in electrical engineering from the University of New Mexico, where he became affiliated with the NASA Center for Autonomous Control Engineering (ACE). His dissertation addresses adaptive hierarchical control of fuzzy behavior-based systems and its application to rover navigation. In his current capacity as a Robotics Engineer he develops autonomous control and navigation technology for planetary rover research and flight missions. His research interests include mobile robot navigation, autonomous control, cooperative robotics, and applications of fuzzy logic and other soft computing techniques to intelligent control systems.



Mohammad Jamshidi received the Ph.D. degrees from the University of Illinois at Urbana-Champaign in February 1971. He holds an honorary doctorate degree from Azerbaijan National University, Baku, Azerbaijan, 1999. Currently, he holds the AT&T Professorship of manufacturing engineering, Professor of Electrical and Computer Engineering and founding Director of Center for Autonomous Control Engineering (ACE), and founding Director of Computer-Aided Design Laboratory for Intelligent and Robotic Systems at the University of New Mexico, Albuquerque, NM, USA. He is also a member of the NASA National Board for Minority Small Businesses Utilization and US National Academy of Sciences National Research Council's Integrated Manufacturing Review Board. In 1994–95 academic year, he was a Directeur de Recherche Associe' at Laboratoire d'Analyse et d'Architecture des Systems (LAAS) of Centre National de Recherche Scientifique (CNRS) in Toulouse, France. Previously he spent 6 years at US Air Force Phillips (formerly Weapons) Laboratory working on large-scale systems and control of optical systems and adaptive optics. He has worked in various academic and industrial positions at the University of Illinois, Urbana, IL; Shiraz (formerly Pahlavi) University, Shiraz, Iran; University of Stuttgart, Stuttgart, Germany; IBM Research Division, Yorktown Heights, New York; Technical University of Denmark, Lyngby, Denmark; IBM information Products Division, Boulder, Colorado; General Motors Research Laboratories, Warren, MI; George Washington University, Washington, DC; National Institute of Standards and Technology (formerly NBS), Gaithersburg, MD; University of Virginia, Charlottesville, VA. He has close to 430 technical publications including 42 books and edited volumes. Six of his books have been translated into at least one language. He is the Founding Editor or co-founding editor of 5 journals (including Elsevier's *International Journal of Computers and Electrical Engineering*) and one magazine (*IEEE Control Systems Magazine*). He has been on the executive editorial boards of a number of journals and two encyclopedias. He is the series editors for ASME Press Series on *Robotics and Manufacturing* since 1988 and Prentice Hall Series on *Environmental and Intelligent Manufacturing Systems* since 1991. In 1986 he helped launch a specialized Symposium on robotics which was expanded to International Symposium on Robotics and Manufacturing (ISRAM) in 1988, and since 1994 it was expanded into World Automation Congress (WAC) where it now encompasses five main symposia and forum on Robotics, Manufacturing, Automation, Control, Soft Computing, Multimedia and Image Processing. He has been the General Chairman of WAC from its inception. Dr. Jamshidi is a Fellow of IEEE, an Associate Fellow of Third World Academy of Sciences (Trieste, Italy), Associate Fellow, Hungarian Academy of Engineering, Corresponding Member, Persian Academy of Engineering, member of New York Academy of Science, member of American Association for the Advancement of Science (AAAS), and recipient of the IEEE Centennial Medal and IEEE Control Systems Society Distinguished Member Award, member of five honor societies and honorary chaired professor at 3 Chinese universities. He is a Consulting Editor of the EOLSS — Encyclopedia of Life Support Systems — a global scientific project for the next century, headquartered in London, UK. He is the founder and now Vice President of TSI Enterprises, Inc. — a 15-year old New Mexico Corporation specializing in educational and research tools on intelligent systems techniques for automation and control as well as publishing and multimedia.