Training of Fuzzy-Neural Network for Voice-Controlled Robot Systems by a Particle Swarm Optimization

Keigo Watanabe[†], Amitava Chatterjee[†], Koliya Pulasinghe[†],
Sang-Ho Jin[‡], Kiyotaka Izumi[†] and Kazuo Kiguchi[†]

[†] Department of Advanced Systems Control Engineering, Saga University, 1-Honjomachi, Saga 840-8502, Japan
(Tel: +81-952-28-8602; E-mail: {watanabe, izumi, kiguchi}@me.saga-u.ac.jp)
[‡]Department of Mechanical Engineering, Doowon Technical College, 678, Jangwon-ri, Juksan-myon,
Ansung-shi, Kyonggi-do, 456-718 Korea
(Tel: +82-31-670-7134; E-mail: shjin@doowon.ac.kr)

Abstract: The present paper shows the possible development of particle swarm optimization (PSO) based fuzzy-neural networks (FNN) which can be employed as an important building block in real life robot systems, controlled by voice-based commands. The PSO is employed to train the FNNs which can accurately output the crisp control signals for the robot systems, based on fuzzy linguistic spoken language commands, issued by an user. The FNN is also trained to capture the user spoken directive in the context of the present performance of the robot system. Hidden Markov Model (HMM) based automatic speech recognizers are developed, as part of the entire system, so that the system can identify important user directives from the running utterances. The system is successfully employed in a real life situation for motion control of a redundant manipulator.

Keywords: Particle-swarm optimization, fuzzy-neural network, voice-controlled robots, redundant manipulators

1. Introduction

Particle swarm optimization (PSO) has very recently emerged as an important combinatorial metaheuristic technique for both continuous-time and discrete-time optimization. In the last ten years or so, many research efforts have been directed towards development and exploration of this new technique which is based on the social metaphor of bird flocking or fish schooling [1]–[3]. Like genetic algorithm (GA), PSO is also initialized with a random population of candidate solutions which are flown in the multidimensional search space in search of the optimum solution [4],[5]. One of the main advantages of PSO, which has endeared itself to the research community is that it is comparatively simple in operation and easier to understand compared to other evolutionary computations presently available, e.g. GA, evolutionary programming, genetic programming, etc. [4]. PSO employs smaller number of free, tunable parameters [5] and hence should be able to attract more attention from the industrial community which would prefer a simple yet efficient algorithm to solve their relevant problem domains.

The early works on PSO have shown the employment of the algorithm for a number of benchmark problems with a variety of dimensions. Most of these experimentations indicated that PSO can be very useful in certain problem domains to arrive at a fast solution [5]. To overcome getting stuck in local minima, different improved variations of PSO have been reported which additionally employ static and varying inertia weights and constriction factor [2]. PSO has also recently evolved as a viable alternative to tune fuzzy and neuro-fuzzy systems [7]. However one of the main question that still remains unanswered is that can PSO be effectively applied in developing a real world system? This question inspired us to undertake the present work.

The present paper describes the effective utilization of PSO to train a Takagi-Sugeno (TS)-type fuzzy-neural network (FNN) as an important building block of voice-controlled robot systems. Voice-controlled robots are gradually attracting more and more attention due to their capability of incorporating human-friendly spoken, natural language based interface [10]. Development of socially responsible, mature Robots guided by spoken language commands can be very useful for nursing and aiding the elderly people, for physically handicapped people, for people struck with paralyses and even as companion for children. The PSO has been successfully employed to tune the TS-type FNN which is employed to acquire fuzzy linguistic information from human experts and to provide crisp and smooth performance for a selected action of the robot. The suitability and effectiveness of the proposed PSO-trained FNN for voice-controlled robot systems are aptly demonstrated by applying it for a real life situation for motion control of a seven degrees-of-freedom redundant robot manipulator.

Amitava Chatterjee was on leave from Electrical Engineering Department, Jadavpur University, Kolkata - 700 032, India, when this work was performed.

2. The Particle Swarm Optimization

PSO, like other evolutionary computations, always initializes a pool of particles with random positions and velocities in a multidimensional space. The algorithm is shown in form of a flow chart in Fig. 1. In each iteration k PSO calculates the fitness function f for each potential solution, by utilizing the current positional coordinates of the jth particle $\boldsymbol{x}_j$ in N-dimension. If the value of the fitness function, f, is not found satisfactory, then the position $\boldsymbol{x}_j$ and velocity $\boldsymbol{v}_j$ of the jth particle is updated according to position and velocity update relations. The new velocity of the jth particle in nth dimension, for the $(k+1)$th iteration, is calculated as an additive influence of three major components, i) component *I*: the current velocity (at kth iteration) of the jth particle in nth dimension (denoted by v_{jn}), ii) component *II*: the difference between the nth dimension component of the best position obtained by the jth particle, until now (denoted by p_{jn}) and the current position (at the kth iteration) of the jth particle in nth dimension and iii) component *III*: the difference between the nth dimension component of the best position obtained by any particle in the topological neighborhood of the jth particle, until now (denoted by p_{gn}) and the current position (at the kth it-

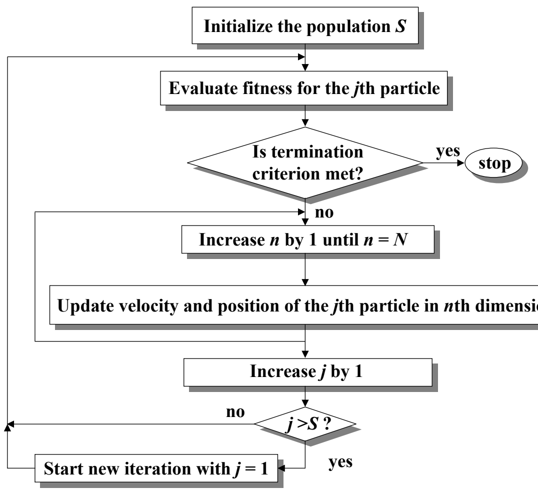
Initialize the population *S*
Evaluate fitness for the *j*th particle
Is termination criterion met?
yes
stop
no
Increase *n* by 1 until *n* = *N*
Update velocity and position of the *j*th particle in *n*th dimension
Increase *j* by 1
no
j >*S* ?
yes
Start new iteration with *j* = 1

Fig. 1. The Particle Swarm Algorithm.

eration) of the jth particle in nth dimension. The influence of each of the components II and III are stochastically weighted and added to component I to obtain the updated velocity. This velocity update relation can be given as:

$$\begin{aligned} v_{jn}(k+1) &= v_{jn}(k) + \varphi_1(p_{jn}(k) - x_{jn}(k)) \\ &\quad + \varphi_2(p_{gn}(k) - x_{jn}(k)) \end{aligned} \tag{1}$$

φ_1 and φ_2 are two uniformly distributed random positive numbers, used to provide the stochastic weighting, and they are restricted by the maximum value of $\varphi_{\max}$. Usually $\varphi_{\max}$ is chosen between 1.6 and 2.0. To prevent any unwanted exploration of a particle along a given dimension, its velocity in that dimension is restricted by its maximum permissible value ($v_{n\max}$). This option is exercised to keep the random search of the potential solutions, in quest of a better fitness, within control. Then the new position of the jth particle in nth dimension is calculated as:

$$x_{jn}(k+1) = x_{jn}(k) + v_{jn}(k+1). \tag{2}$$

However, like many other evolutionary computations, PSO algorithm with these velocity and position update rules also suffers from the problem of too aggressive search in the problem space when $v_{n\max}$ is chosen high. This can cause large oscillations in the trajectories of the particles prohibiting them to settle to a reasonable solution due to too course tuning in each iteration. On the other hand, choice of a too small value for $v_{n\max}$ can cause very small updating of velocities and positions of particles in each iteration. Hence the algorithm may take a long time to converge and faces the problem of getting stuck to local minima. To overcome these situations various researchers have recently proposed improved velocity update rules with dynamic inertia weights or employing constriction coefficients. Velocity update relations with dynamic inertia weights can be given as:

$$\begin{aligned} v_{jn}(k+1) &= W(k+1)v_{jn}(k) + \varphi_1(p_{jn}(k) - x_{jn}(k)) \\ &\quad + \varphi_2(p_{gn}(k) - x_{jn}(k)). \end{aligned} \tag{3}$$

Here the influence of the old velocity on the new velocity is weighted by the inertia weight W. This inertia weight is usually chosen a variable quantity. For early iterations, they can be chosen

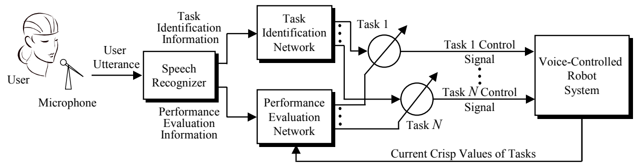
User
Microphone
User Utterance
Speech Recognizer
Task Identification Information
Performance Evaluation Information
Task Identification Network
Performance Evaluation Network
Task 1
Task *N*
Task 1 Control Signal
Task *N* Control Signal
Voice-Controlled Robot System
Current Crisp Values of Tasks

Fig. 2. The architecture of the proposed voice-controlled robot systems.

higher so that the particles are allowed to have much exploration capability to aggressively search the solution space. Once the algorithm is found to converge more and more towards the optimum, this coarse tuning is gradually converted to a finer and finer tuning by making W smaller in later iterations. This provides the provision of improving accuracy of the optimization algorithm. As opposed to the proposal of incorporating inertia weights which only exercise its influence over component *I* of the velocity update relation in (1), constriction coefficients are employed to exercise wider control over each of the three major components of the velocity update relation in (1), in an effort to prevent explosion of the system. The proposed velocity update relation can be given as:

$$\begin{aligned} v_{jn}(k+1) &= \chi(v_{jn}(k) + \varphi_1(p_{jn}(k) - x_{jn}(k)) \\ &\quad + \varphi_2(p_{gn}(k) - x_{jn}(k))). \end{aligned} \tag{4}$$

It has been argued that there is no requirement of restricting velocity in any dimension for PSO employing constriction coefficient. The constriction coefficient χ makes this requirement redundant. However it has also been pointed out that employing constriction coefficient with a liberal $v_{n\max}$ equal to the dynamic range of the variable may be quite useful [5].

3. Fuzzy Neural Network in Voice-Controlled Robot Systems

The proposed voice-controlled robot system is shown in form of a schematic diagram in Fig. 2. The system is composed of four major building blocks: a speech recognizer (SR), a task identification network (TIN), a performance evaluation network (PEN) and the robot system under consideration. The input to the system appears in form of spoken commands from a human. The entire system is designed so that the linguistic nature of spoken directive of the human user is translated in form of a quantified and crisp desired action for the robot system. To start with, each and every running utterance from an user is stripped off by a speech recognizer (SR) module to create a pseudo sentence. Creating a pseudo sentence from an user utterance implies that the in-vocabulary (IV) words are segregated from out-of-vocabulary (OOV) words. For example, if an user utters *"Robot, can you move backward very slow"*, then the OOV words *Robot*, *can* and *you* are stripped off to create the pseudo sentence *"move backward very slow"* comprising of IV words. This pseudo sentence actually consists of the meaningful semantic action along with its linguistic adjective for the robot to perform its action. Spotting of proper IV words in a running utterance is achieved by training a left-to-right category of Hidden Markov Models (HMMs) [8],[9]. The construction of this pseudo sentence is achieved by using the HMM Toolkit (HTK) distributed as a freeware in the web by Speech Vision and Robotics Group of the Engineering Department, Cambridge University. This speech recognizer employs phoneme based recognition of IV words where the phonemes of each IV word are designed as tri-state left-to-right HMM models. The strength of a keyword recognizing module depends on how intelligently the IV words can be recognized and it requires efficient training of the HMM based speech recognizer. Special attention has been provided to identify similar actions with synonyms. Initially HMMs are trained so that they can identify isolated words and later they are added together to build the database for the SR.

Once the pseudo sentence is constructed, it comprises of two parts: each definite task/action along with the evaluation of performing that task. For example, in the pseudo sentence *"move backward very slow"*, the task/action part is *"move backward"* and the evaluation of performance is *"very slow"*. The selection of the task part comprises of the qualitative nature of the job and performance evaluation part is executed to determine the quantitative, crisp output signal to be generated by the network for the robot system, for that given task as commanded by the user. For selection of the specific action, the pseudo sentence is fed to a single layer perceptron based artificial neural network (ANN). Here each output node characterizes the presence of a specific IV word in the pseudo sentence by generating a high output. Each output node employs a hard limiting characteristic function and the ANN is overtrained to specifically identify the set of IV words. For a set of M IV words with P distinct tasks the ANN employs M input nodes and P output nodes. Normally $P \leq M$ to take care of different words with similar tasks i.e. synonyms in user utterance.

Once the task identification ANN identifies the exact linguistic nature of the task assigned, it becomes very important to determine the quantitative degree of severity with which the task has to be performed. This very important part of the total system is implemented with the help of the performance evaluation network (PEN). This PEN is implemented with the TS-type FNN which is trained with the help of PSO. This FNN-based PEN is employed to acquire fuzzy linguistic information from the domain expert to train the architecture so that it can produce precise, quantitative control signal for the robot system as an output, based on the user command. The FNN-based PEN utilizes both the linguistic task identification and performance evaluation clauses in the pseudo sentence as its one input. For example in the pseudo sentence *"move backward very slow"*, the linguistic task identification clause *"move backward"* is associated with a fuzzy predicate (FP) symbolizing the performance evaluation clause *"very slow"*. Each linguistic task identification input is associated with an FP which is characterized as a singleton. Each FP belongs to a global FP database for all actions. One sample FP database can be $\{very\, slow,\ slow,\ carry\, on,\ fast,\ very\, fast\}$. The other input of the FNN corresponds to the current quantitative value of the tth task selected from TIN. The architecture of the FNN is shown in Fig. 3.

3.1. Architecture of the TS-type FNN

The first three layers of this six-layered FNN correspond to the *antecedent* parts and the last two layers of the FNN correspond to the *consequence* parts of the fuzzy reasoning. As shown in Fig. 1, the domain of discourse of task 1 is described by fuzzy variable A^1 with q number of linguistic values and the same of action N is described by fuzzy variable A^N with t number of linguistic values. Similarly, the FPs associated with task 1, FP^1s, consist of r number of FPs and those of FP^Ns consist of s number of FPs. Thus, each action is unique in the sense of domain of discourse and FPs associated with that particular action. We employ the notations u_i^l and O_i^l as the input to the ith node of lth layer and the output from the ith node of lth layer respectively.

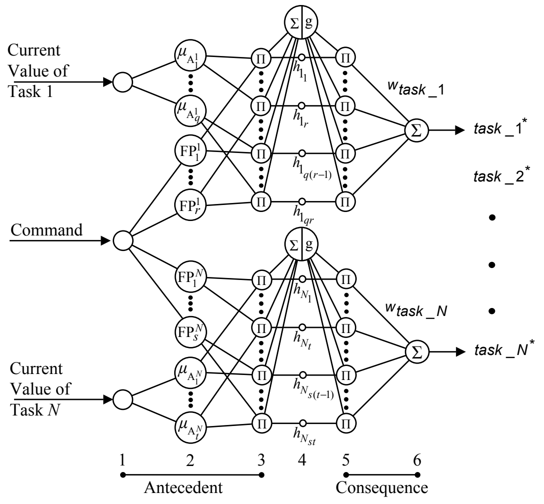

Fig. 3. The FNN employed as the performance evaluation network.

3.1.1 Layer 1: Input layer

Layer 1 consists of two types of nodes: 1) a command node to represent the availability of FPs in the pseudo sentence and 2) normal input nodes for different tasks of the machine which also correspond to the current physical output measured from the robot systems. Each task is labeled as, Current Value of Task i , where $i = 1, \cdots, K$. The current value of task i, i.e., the crisp input to the ith task node is denoted as x_i. The nodes in this layer receive the current values of all machine tasks and FPs of the pseudo sentence and transmit them to the next layer directly. Hence the input-output relation of this layer is given by:

$$O_i^1 = u_i^1 = x_i. \tag{5}$$

3.1.2 Layer 2: Membership function layer

Layer 2 acts as the fuzzification layer of the FNN where the values of the activated fuzzy membership functions (MFs) for a given current value of task i are calculated. Let us consider that the current value of the ith task x_i has activated the vth *linguistic value* or *MF*, A_v^i, of the *linguistic variable* A^i associated with task i. Then the input-output relation of this layer, for these nodes, is given by:

$$O_{iv}^2 = f(O_i^1) = f(u_i^1) = \mu_{A_v^i}\left(u_i^1\right) = \mu_{A_v^i}\left(x_i\right). \tag{6}$$

The outputs of the FP nodes are activated like fuzzy singletons. An FP node connected to a command node will produce an output of 1 if the FP exists in the pseudo sentence. Let the vth FP for the ith task activated by the pseudo sentence be denoted by FP_v^i. Then the output from such a node can be obtained as:

$$O_{ij}^2 = f(u_i^1) = f(FP_v^i) = \begin{cases} 1 & \text{if FP exists in the command} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

3.1.3 Layer 3: Rule layer

The rule layer employs AND operation, in form of algebraic product, to calculate the T-norm of the antecedent part. This layer essentially computes the strength with which each rule can be fired or activated. Computation of the T-norm employing algebraic product ensures that only those rules which contain fuzzy predicates in the pseudo sentence will be activated. These rule values, which represent the user request and the current status of the machine, are exploited in layer 5 for normalization purpose. The output of the rth rule for the ith task from this layer is obtained as:

$$O_{ir}^3 = h_{i_r} = \mu_{A_p^i}(x_i) * FP_q^i \tag{8}$$

where h_{i_r} represents the firing strength of the rth rule of the ith task. The total number of rules that can be activated associated with this task is R where $R = P \times Q$. Here P and Q are the total number of MFs of the linguistic variable and the total number of fuzzy predicates associated with the ith task.

3.1.4 Layer 4: Intermediate computation layer

The first node of layer 4 of each FNN, assigned for each specific task, performs an integration function followed by an inverse function. This is shown by the symbols $\sum$ and g respectively. The output from this node of layer 4, for task i, can be obtained as:

$$O_{i1}^4 = g(z) = \frac{1}{z} \tag{9}$$

where z is obtained as:

$$z = \frac{1}{\sum_{r=1}^{R} u_{ir}^4} = \frac{1}{\sum_{r=1}^{R} h_{i_r}}. \tag{10}$$

Outputs from the other nodes m in this layer, for task i, are obtained as:

$$O_{im}^4 = u_{im}^4. \tag{11}$$

3.1.5 Layer 5: Normalization layer

This layer performs the normalization function for the crisp values of the activated output rules. The output from each mth node in each FNN, i.e. for the task i, in this layer is obtained by multiplying the input for the mth node and the output of the first node from layer 4, for the task i. Then the output of any mth node of this layer can be given by:

$$O_{im}^5 = u_{im}^5 * O_{i1}^4 = \frac{h_{i_m}}{\sum_{r=1}^{R} h_{i_r}}. \tag{12}$$

3.1.6 Layer 6: Defuzzification layer

Layer 6 is the defuzzification layer of the FNN. Each output node in this layer performs defuzzification to generate final, crisp output for each specific action, by taking weighted average of all its inputs. The final output is dependent on the inputs as well as the connecting weights between the inputs and the output. The overall output, i.e., the quantitative evaluated performance value for the ith desired task is given as:

$$O_i^6 = task\_i^* = \frac{\sum_{r=1}^{R} h_{i_r} w_{ir}}{\sum_{r=1}^{R} h_{i_r}}. \tag{13}$$

Here w_{ir} denotes the connecting weight for the rule r to perform the desired task i, i.e., firing strength of the rule r.

3.2. Training of the TS-type FNN employing PSO

The training algorithm of the TS-type FNN based PEN is shown in Fig. 4. Here the PSO algorithm has been employed to train the weights w_{ir} in the defuzzification layer i.e. layer 6 of the FNN architecture. The PSO problem has been defined as an N-dimensional problem where N is the total number of output weights in the FNN-based PEN. Our objective is to train the output weights of the FNN so that the PEN can produce desired crisp output control signal according to the pseudo sentence derived from the spoken directive from the user and the current state of the robot task. The FNN is trained in batch mode with a training dataset of I data pairs. The problem is formulated as a minimization problem where the fitness function is based on the mean-squared-error (MSE) of the FNN in each iteration, given as:

$$MSE = \frac{1}{I}\sum_{i=1}^{I}(y_{di} - y_{ai})^2. \tag{14}$$

Here y_{di} is the desired output and y_{ai} is the actual output from the FNN for the task i in the training phase. The desired output is obtained from a knowledge base, created using the knowledge of the domain expert for that specific robot system problem domain. All the training data pairs are created by employing completely randomly chosen numbers for each input. Then, we choose a possible size of population S which gives us a possible set of weight vectors $\{\boldsymbol{w}_1 \ \boldsymbol{w}_2 \ \cdots \ \boldsymbol{w}_S\}$. Each weight vector $\boldsymbol{w}_j$ is a potential particle for the PSO algorithm and is an $R \times 1$ vector where R is the total number of output weights in the FNN. Hence $\boldsymbol{w}_j = [w_{j1} \ w_{j1} \ \cdots \ w_{jR}]$ where $R = P \times Q$. To start with, all the weights in each weight vector $\boldsymbol{w_j}$ are randomly initialized in a discourse of $[w_{left}, w_{right}]$. Then the FNN is fed with it's dataset in batch mode with each possible potential solution (i.e. weight vector $\boldsymbol{w}_j$) and error is calculated for each data pair in the dataset. When the entire dataset is presented to FNN for once, we calculate MSE of the system, for the given $\boldsymbol{w}_j$. In this process we evaluate the performance of the FNN for the entire dataset for each possible weight vector $\boldsymbol{w}_j$. If, in each case, it does not meet the termination criterion, i.e. it is not lower than the maximum permissible MSE, $MSE_{\max}$, then it can be concluded that none of the possible particles i.e. weight vectors, can give satisfactory solution and that completes one iteration of the optimization process. Then, in next iteration, position coordinates of each of these particles, i.e. numerical values of each component weight in each possible weight vector, are updated according to the PSO algorithm. At the end of the kth iteration, the nth dimension i.e. the nth component weight w_{jn} in the jth particle, i.e. the possible weight vector $\boldsymbol{w}_j$ is updated according to the formulae,

$$\begin{aligned}\dot{w}_{jn}(k+1) &= W(k+1)\dot{w}_{jn}(k) + \varphi_1(p_{jn}(k) - w_{jn}(k)) \\ &\quad + \varphi_2(p_{gn}(k) - w_{jn}(k))\end{aligned} \tag{15}$$

and

$$w_{jn}(k+1) = w_{jn}(k) + \dot{w}_{jn}(k+1). \tag{16}$$

In our version of the PSO we have employed a linearly adaptable inertia weight W which starts with a high value W_{high} and linearly

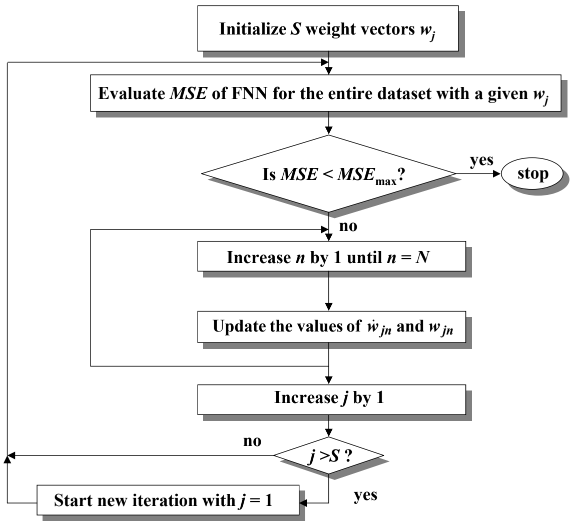
Initialize S weight vectors w_j
Evaluate MSE of FNN for the entire dataset with a given w_j
Is $MSE < MSE_{\max}$?
yes
stop
no
Increase n by 1 until $n = N$
Update the values of $\dot{w}_{jn}$ and w_{jn}
Increase j by 1
no
$j > S$?
yes
Start new iteration with $j = 1$

Fig. 4. The training of the FNN employing PSO.

decreases to W_{low} at the end of the maximum number of iterations, i.e. $iter_{\max}$, if the algorithm continues till the end without meeting the termination criterion. Hence inertia weight for the $(k+1)$th iteration is computed as:

$$W(k+1) = W_{low} + \left(\frac{W_{high} - W_{low}}{iter_{\max}}\right)(iter_{\max} - (k+1)). \tag{17}$$

If the optimization routine converges successfully before $iter_{\max}$ is reached, then definitely W will end up with a value higher than W_{low}. For each dimension we can choose different discourse of W i.e. $[W_{high},\ W_{low}]$, if we wish so. Similarly the maximum permissible velocity of each particle i.e. the maximum permissible rate of change of each weight, $\dot{w}_{\max}$, can also be accordingly chosen independent of each other.

One of the significant features of building the knowledge base from the domain expert and utilizing it in the training phase of the FNN is the contextual meaning of each word spoken by the user. The importance of each linguistic information in the pseudo sentence is fuzzily converted in the context of the present situation of the robot system and the new control signal is accordingly generated. For example let us consider the pseudo sentence *"move forward very slow"*. Then if the robot is already moving forward with its velocity very near its minimum velocity, the importance of the new linguistic directive *"very slow"* will be at its lowest and the decremental change in velocity will also be smallest. However, if the robot is already moving forward with its velocity very near its maximum velocity, then this linguistic directive will attain its maximum importance and the decremental change in velocity will achieve its greatest value.

4. Motion Control of a Redundant Manipulator

A real life experimental case study was conducted to control the motion of a seven degrees-of-freedom (DOF) redundant manipulator employed to perform a real assembly task. Figure 5 shows the layout of this assembling task undertaken. They are mainly employed to perform tasks that are considered difficult or impossible for regular manipulators, e.g. tracking a desired trajectory in presence of obstacles and/or minimizing a performance objective, defined as a

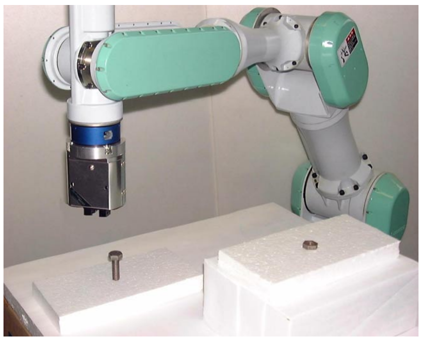

Fig. 5. The layout of the assembly task for PA 10.

function of motion [11],[12]. The redundant manipulator employed in our experiment is the PA-10 portable general purpose arm, from Mitsubishi. Our system employs Via Voice SDK, supplied by IBM, to develop the automatic speech recognizer (ASR) [13]. Here the grammar for identifying words is defined with the help of SRAPI recognition control language (SRCL) syntax in plain-text Backus Naur Form (BNF). The entire pool of IV words is developed by considering three sub-pool of IV words, dedicated for three specific operations: 1) to manipulate the gripper tool, 2) to rotate the wrist clockwise and anticlockwise and 3) to control the traversal of the end effector in a three-dimensional space. The SR is trained for this entire pool of IV words by pronouncing each word separately so that these words get added to the existing pronunciation pool of the ASR.

Here the input fuzzy variable for the FNN-based PEN, which is identical with the output sensed from the redundant manipulator, is taken as linear distance. This is required for training the FNN to achieve desired traversal of Euclidean distance by PA 10 for precise target positioning. The target position is described with the help of the three-dimensional Cartesian coordinate system. Figure 6 shows the three MFs chosen for this fuzzy variable, input distance. We chose four fuzzy singletons for this variable. Hence, the defuzzification layer for the FNN contains 12 output weights which are trained by employing PSO in the training phase. Hence the PSO problem is formulated as the minimization of MSE problem with 12 dimensions. The free parameters of the PSO are chosen as W_{high} = 0.2, W_{low} = -0.3 and $\dot{w}_{\max}$ = 0.1 for each of the 12 dimensions. Initially all particles in each dimension i.e. all the possible weight values are initialized in the domain $[0, 1]$. Here also we tried to solve the PSO problem with three possible population sizes, i.e. 20, 30 and 40. Figure 7 shows the training performance of each of these PSO algorithms. Here PSO showed satisfactory convergence phenomenon when the population size was chosen as 40. For smaller population size, the performance in the training phase of the FNN-based PEN was found unsatisfactory. We implemented the real life FNN-based PEN utilizing the output weights trained by the PSO, employed with 40 particles. During the training process of the FNN, the knowledge base designed according to the knowledge acquired from the domain expert is given in tabular form in Table 1.

Figure 8 shows the results obtained in assembling a bolt to a nut, using PA 10, which were shown in the experimental setup in Fig. 5. The spoken language directives employed by the user, required for successful completion of this task, along with the corresponding tip positions of the manipulator in three-dimensional Cartesian co-

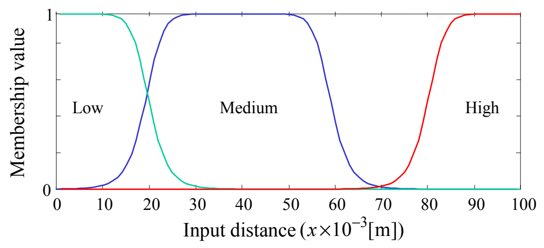

Fig. 6. Membership functions of input distance for PA 10.

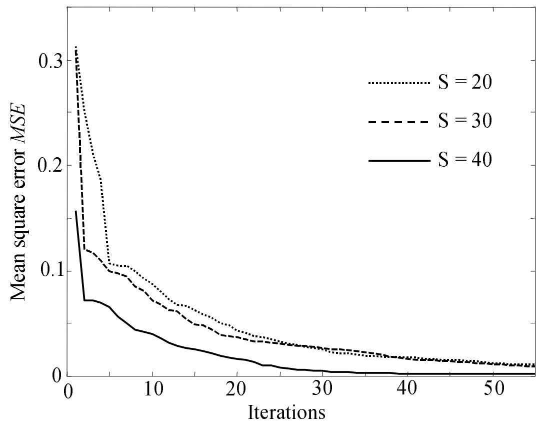

Fig. 7. The training performance of PSO for PA 10.

ordinate system are shown in Table 2. A successful completion of the specified job demonstrates the efficacy of the FNN based PEN employed for smooth controlling of motion of the PA 10 arm along with its end effector.

5. Conclusions

The present work has demonstrated the feasibility of employing particle swarm optimization (PSO) techniques for efficient training of a fuzzy-neural network (FNN). The PSO trained FNN has been successfully employed as an important building block in real life voice-controlled robot systems. When properly trained by PSO, the FNN can perform accurate controlling of the robot behavior on the basis of fuzzy linguistic commands issued by an user, in form of natural, spoken language based running utterances. The system has been developed so that it is sensitive to the context of the spoken language based directive from the user, when observed from the perspective of the end-effector. The system has been successfully employed to control the motion of a redundant manipulator for an assembly task, also performed in real life.

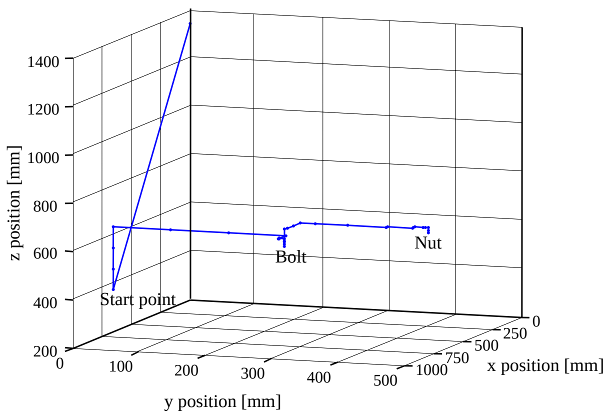

Fig. 8. The profile of the end-effector of PA 10.

Table 1. The knowledge base of the desired distance traversal for PA 10

	Low	Medium	High
A very little	$4.0 + 0.1 \times x$	$4.0 + 0.05 \times x$	$0.1 \times x$
A little bit	$10.0 + 0.1 \times x$	$10.0 + 0.05 \times x$	$0.2 \times x$
Carry on	$0.95 \times x$	$0.95 \times x$	$0.85 \times x$
A far	$40.0 + 1.1 \times x$	$10.0 + 1.1 \times x$	$1.1 \times x$

Table 2. Task related information and corresponding tip positions of the PA 10 manipulator

Task information	x	y	z
move to start position	657.97	0.06	374.57
move a far up	657.94	0.14	460.82
carry on, carry on	657.92	0.13	548.14
repeat again	658.06	0.14	635.42
proceed to far left	658.03	87.14	635.46
carry on, carry on	658.04	174.00	635.43
go a far left	658.02	261.06	635.40
come forward	722.01	261.07	635.46
go a very little back	718.30	261.07	635.44
move a very little back	714.35	261.06	635.47

References

[1] Y. Shi and R. C. Eberhart, “Empirical study of particle swarm optimization,” in *Proc. 1999 Congr. Evolutionary Computation, IEEE Service Center*, Piscataway, NJ, 1999, pp. 1945–1950.

[2] R. C. Eberhart and Y. Shi, “Comparing inertia weights and constriction factors in particle swarm optimization,” in *Proc. 2000 Congr. Evolutionary Computation*, San Diego, CA, July 2000, pp. 84–88.

[3] J. Kennedy, “The particle swarm: Social adaptation of knowledge,” in *Proc. 1997 Int. Conf. Evolutionary Computation*, Indianapolis, IN, April 1997, pp. 303–308.

[4] P. Angeline, “Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences,” in *Evolutionary Programming VII*, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 601–610.

[5] M. Clerc and J. Kennedy, “The particle swarm-explosion, stability, and convergence in a multidimensional complex space,” *IEEE Trans. Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[6] A. Conradie, I. Nieuwoudt, and C. Aldrich, “Nonlinear neurocontroller development with evolutionary reinforcement learning,” in *9th National Meeting of SAIChe*, Secunda, South Africa, 2000.

[7] A. Conradie, R. Miikkulainen, and C. Aldrich, “Adaptive control utilising neural swarming,” in *Proc. Genetic and Evolutionary Computation Conference*, New York, USA, 2002.

[8] L. R. Rabiner and B. H. Juang, “An introduction to hidden Markov models,” *IEEE ASSP Magazine*, vol. 3, pp. 4–16, Jan. 1986.

[9] R. C. Rose and D. B. Paul, “A hidden Markov model based keyword recognition system,” in *Proc. IEEE ICASSP ’90*, 1990, pp. 129–132.

[10] K. Pulasinghe, K. Watanabe, K. Kiguchi, and K. Izumi, “Modular fuzzy neuro controller driven by voice commands,” in *Proc. ICCAS 2001*, 2001, pp. 194–197.

[11] M. C. Ramos, Jr. and A. J. Koivo, “Fuzzy logic-based optimization for redundant manipulators,” *IEEE Trans. Fuzzy Systems*, vol. 10, no. 4, pp. 498–509, Aug. 2002.

[12] Y. Nakamura and H. Hanafusa, “Optimal redundancy control of robot manipulators,” *Int. J. Robot. Res.*, vol. 6, no. 1, pp. 32–42, 1987.

[13] IBM(R) ViaVoice(tm) SDK, “SMAPI Developer’s Guide March 2001,” *International Business Machines Corporation, USA*, 1999–2001.