Robot Inverse Kinematics via Neural and Neurofuzzy Networks: Architectural and Computational Aspects for Improved Performance

SPYROS N. RAPTIS, ELPIDA S. TZAFESTAS, and SPYROS G. TZAFESTAS

Intelligent Robotics and Automation Laboratory (IRAL)
Department of Electrical and Computer Engineering
National Technical University of Athens
Zographou Campus, 15773, Athens, GREECE
***Tel**.: +30-1-772 2489, **Fax**: +30-1-7722490, **E-mail**: tzafesta@softlab.ece.ntua.gr*

Abstract: *Neural networks are universal nonlinear-function approximators and have for long time been used to implement various practical nonlinear inverse mappings. The choice of network type and structure depends on the mapping type and the degree of generalization required. The use of neural networks for solving the inverse robot kinematics has been extensively studied by many workers, but still some problems related to the complexity and strong nonlinearity of the inverse kinematics process need suitable heuristic and ad-hoc techniques and simplifications. The aim of this paper is exactly to contribute towards filling this gap by investigating a number of computational and architectural issues so as to improve the performance of the implementation of the inverse kinematics process. These issues include the generation and preprocessing of training data, the data scaling, the treatment of multiple solutions, the reduction of the approximation error, and the speeding-up of the training process. A generic neural network architecture is proposed which employs multiple radial-basis function (RBF) network elements and the "mixture of local experts" principle. An algorithm is presented for the training data preprocessing which greatly reduces the training time and overall system error. Two fuzzy-logic solutions are provided and discussed; one employing a fuzzy associative memory (FAM), and the other a neurofuzzy cell architecture proposed by the authors. In all cases no knowledge is assumed about the inverse kinematics of the robot at hand, as long as its forward kinematics is known. Some indicative simulation results are included and discussed.*

Keywords: *Neural networks, inverse kinematics, radial basis function networks, data preprocessing, fuzzy associative memory, neurofuzzy cell.*

1. Introduction

Robotics is a research and development field with diverse applications in manufacturing, hazardous environments, aerospace, undersea research, medical services and domestic households. Industrial robotic manipulators are complex nonlinear dynamic systems, typically modeled as a serial chain of *n* rigid bodies (links). One end of the chain is usually fixed to some reference surface (frame), whereas the other end is free, thus constituting an open kinematic chain of moving rigid bodies. The three main problems in industrial robotic research are: kinematics, dynamics, and control. Kinematics is distinguished in forward kinematics and inverse kinematics. The forward kinematics problem is a simple and

straightforward problem consisting primarily of finding the position and orientation of the end effector in a Cartesian space given particular joint angles. Inverse kinematics deals with finding the joint angles for given position and orientation of the end effector in the Cartesian space, and can be very complex depending on the robot structure and number of degrees of freedom. The inverse kinematics problem has usually many possible solutions and it is not always obvious which set of joint angles to select.

Although in many cases there exists a closed form solution, there are many others which need time-consuming iteration that makes it not suitable for real-time application. Therefore an attempt was made by many workers to eliminate coordinate transformation from the Cartesian to the joint space for robot control applications. One solution is to compute the joint torques from the desired Cartesian trajectory [25,26], and another is to solve for the joint velocities instead of the joint angles [27] which is an easier task. The alternative approach is to use universal approximators based on neural, fuzzy and neuro-fuzzy learning [1-3, 13, 17, 28-37].

The purpose of the present paper is to investigate and treat some of the key problems encountered when solving the inverse kinematics through neural and neuro-fuzzy learning, namely the problems of generating and preprocessing training data, handling multiple solutions, reducing the approximation error, and lowering the training time. To this end, a general neural architecture is proposed which employs multiple radial basis function (RBF) networks and is based on the "mixture of local experts" paradigm, and an algorithm is presented for the preprocessing of the training data which reduces considerably the training time and the final overall system error. Regarding the solution through fuzzy logic, a fuzzy associative memory (FAM) is trained with the aid of the forward dynamic equations, so as to map the inverse kinematics solution. No knowledge about the inverse kinematics of the robot is required as long as its forward kinematics is known.

Section 2 discusses the inverse kinematics problem and the conventional approaches to handle it. Section 3 presents the two basic alternative ways for solving the inverse kinematics problem via neural networks, namely: the function approximation approach and the associative memory approach. Section 4 provides a review of the basic features of neural inverse kinematics (nonuniformity of training data distribution, multiple solutions, learning issues). Section 5, which is the core of the paper, presents the solutions to the various subproblems faced when employing the neural inverse kinematics approach. Section 6 deals with the fuzzy and neuro-fuzzy approach to robot inverse kinematics. Two solutions are investigated. The first uses a fuzzy associative memory, and the second a neuro-fuzzy cell structure suitable for general fuzzy inference. Section 7 provides an outline of the results obtained through neural and neurofuzzy inverse kinematics. Finally, Section 8 gives the conclusions.

2. The Robot Inverse Kinematics Problem: Statement and Conventional Approach

The study of the kinematic behavior of a manipulator involves the forward (direct) and inverse kinematics analyses. *Forward kinematics*, deals with transforming joint values (coordinates in joint space) into Cartesian coordinates of the end-effector (or the tool center point). For a manipulator of *m* degrees of freedom (DOFs), i.e. of *m* revolute or translational joints, the forward kinematics equation can be written as:

$$p(t) = f\left(q(t)\right) \tag{1}$$

where q is an m-vector of joint values, p is an n-vector of end-effector coordinates, and $f(\cdot)$ is continuous nonlinear function that depends on the known kinematic parameters of the robot at hand. On the other hand, the *inverse kinematics* problem deals with the inverse mapping:

$$q(t) = f^{-1}\big(p(t)\big) \tag{2}$$

The most direct way to deal with (2) is to obtain a closed form solution from (1), but for many robots, this is not possible due to the complex and nonlinear nature of f.

Another approach for addressing the inverse kinematic problem is to use the linear relation between the joint and Cartesian velocities:

$$\dot{p}(t) = J(q)\dot{q}(t) \tag{3}$$

where $J(q)$ is the $n \times m$ Jacobian matrix and can, in general, be singular. This way explicit calculation of the joint coordinates may be avoided and although a large matrix inversion may be required, this may prove to be simpler.

To determine the joint values for given end-effector coordinates, the joint velocity needs to be computed. This can be done through:

$$\dot{q}(t) = J^{+}\big(q(t)\big)\dot{r}(t) + \Big[I - J^{+}\big(q(t)\big)J\big(q(t)\big)\Big]k(t) \tag{4}$$

where $J^{+}\big(q(t)\big)$ is the pseudoinverse of the Jacobian, I is the identity matrix, and $k(t)$ is an m-vector of arbitrary time-varying variables. The pseudoinverse is clearly crucial for the computation of the joint velocities.

Many analytical and arithmetic methods have been proposed for the solution of the inverse kinematics problem. Among them the *pivot method* that decomposes the pseudoinverse of the Jacobian, J^{+}, into sub-matrices, the *extended pivot method* that directly computes the joint velocities, the *table-lookup method* that calculates J^{+} off-line and stores it in memory, the *residue arithmetic method* that uses a parallel algorithm to compute J^{+}, the *least squares method* that directly computes the joint velocity without explicitly solving J^{+}, etc.

3. Neural Network Approach to Robot Inverse Kinematics

Neural networks have also been used to address the inverse kinematics problem. They are mainly used in two ways, namely for function approximation and as associative memories.

- For *function approximation*, where the neural network is trying to formulate an appropriate input-output mapping based on data obtained by the solution of the *forward* kinematics problem of the robot at hand.
- For *associative memories*, where generic or specific analytical or iterative optimization algorithms exist and are used to produce directly inverse kinematics data for the robot. The neural network is trained in a supervised manner to memorize them and gracefully generalize for unforeseen cases. In this case, the neural network is used to replace these algorithms for the on-line operation. The advantage of the neural network implementation over these algorithms, is that the latter are computationally very expensive and cannot be used in real-time applications, while the neural network can be used as a very fast recalling component, after the learning phase is completed.

Among other methods for achieving neural network based learning control, Barto in [14] specifies the following:

- *Copying an existing controller*, which roughly corresponds to the associative memory perspective as described above;
- *Identification of a system inverse*, which roughly corresponds to the function approximation perspective (e.g. [4]); and
- *Differentiating a model*, (e.g. [11]).

3.1. Neural Function Approximation

In the function approximation perspective, many references are available in the literature that attempt to use a multi-layer perceptron trained with the backpropagation algorithm to solve the inverse kinematics problem, e.g. [13]. Unfortunately, the learning times required prove to be large while the error remains considerably high and contains local bursts.

In [13], the average error was about 5% while the maximum error reached 10%. Therefore, it was concluded by the authors that 'plain' backpropagation multi-layer perceptron neural networks are sufficient only as an initial guess to another iterative inverse kinematics algorithm.

In [17] a neural network structure called dynamic neural processor consisting of relatively complex components called dynamic neural units is presented along with a respective learning algorithm. The model's usefulness for the problem of inverse kinematics is tested on a two-link robot manipulator appropriately constrained so as to avoid multiple solutions.

3.2. Neural Associative Memories

Most of the approaches of this category, make use of the Jacobian and/or its pseudoinverse and of recurrent neural networks (often Hopfield networks) for their implementation.

In [1] a recurrent neural network is designed based on a *reflexive generalized inverse* problem. The methodology relies on recent results on recurrent neural networks for solving matrix equations. A dynamic equation relating J to its pseudoinverse is derived assuming that the Jacobian can be considered to be constant within small enough time intervals during the robot motion. This way, the authors state that no training is required as opposed to the supervised learning neural networks for robot control. A tree DOFs planar robot was used to investigate the feasibility of the proposed method. It is clear that the proposed method does not used neural networks in the conventional way since no training takes place, but only as an algorithm representation scheme, i.e. for storing an input-output mapping which was derived algebraically.

In [2], Hopfield analog neural computation has been proposed to implement the Jacobian control. The states of the neurons represent the converted joint velocities while the network inputs and the connection weights are updated according to the current Cartesian velocity command. But the aim of robot inverse kinematics is to guarantee position tracking rather than velocity tracking and zero steady state velocity as pursued in [2] does not necessarily guarantee position tracking.

In [3] it is argued that the above problem arises due to the fact that the Cartesian position commands are completely ignored in the design (the error criterion is misleading as discussed in a next section). Alternatively, the energy function of the Hopfield network is reformulated to implement a *sliding mode control scheme* that the authors claim to guarantee also position

tracking and enjoys the robustness adherent to sliding mode control combined with the neural networks features.

4. Features of Neural Inverse Robot Kinematics

The application of neural techniques to the inverse kinematics problem of robot manipulators, presents certain special difficulties arising mainly from the ill-posed, nonlinear nature of the problem and the difficulty of effectively inverting a one-to-many mapping. Moreover, the increased solution accuracy required by the robot applications makes the application of neural networks for inverse robot kinematics a real challenge. Some of the main requirements and difficulties of this task are described in the following.

Throughout the rest of the discussion, we will assume that the forward kinematics problem is solved for the robot of interest. This assumption is quite natural since using, for example, the Denavit-Hartenberg (D-H) kinematic parameters and appropriate matrix operations we may obtain the end-effector' s position and orientation (pose) for any set of joint values for any serial open kinematic chain.

In the following we will use the term '*forward model*' to refer to whatever mechanism is available to provide forward kinematic solutions. This, for example, could be the D-H model as proposed by Paul [23], the real robot manipulator equipped with end-effector position detectors [24], a neural network emulator trained for forward kinematics, etc. Moreover, we will assume that this model is *accurate*, i.e. the kinematic parameters are precisely known and the robot of interest is calibrated.

To train a neural network controller for *forward* kinematics, enough inputs may presented to the forward model and the respective outputs obtained. These inputs may be generated at random or uniformly within a certain range of values. The resulting input-output pairs may be fed into a supervised learning algorithm.

For learning the *inverse* mapping using neural networks, the same set of pairs could be used with the inputs exchanged with outputs. In [4], this is called 'general learning'. However, apart from the standard problems of neural training such as the inefficiency of 'plain' backpropagation, the choice of initial weight values etc., at least three more major problems arise during such an endeavor as described in the following.

4.1. Non-Uniformity of Training Data Distribution

Since the data used as input for training the inverse kinematics neural network are the *outputs* of the forward model, they can not be uniformly distributed in the training set. Due to the error criterion used for evaluating network performance (usually sum of squared errors *on the training set*) the output error of the resulting inverse system will be higher for robot responses that are not well represented in the training set.

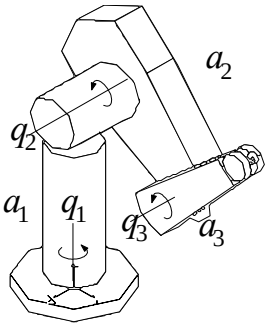

Figure 1. *A typical non-planar PUMA manipulator with 3R lower degrees of freedom.*

For example, consider a generic 3R non-planar manipulator, shown in Fig. 1, which corresponds to the first three lower degrees of freedom of a typical industrial manipulator. Fig. 2 shows the inputs to the forward model as produced by varying each of the joint values with a constant step. These are evenly distributed and are used to produce training pairs. Fig. 3 shows their respective outputs according to the forward model. Since they are interchanged to be fed to the inverse kinematics neural network, it is clear that the network's inputs will certainly *not* be uniformly distributed. Random selection of a subset of these to be the training set may cause some regions of the inputs (Cartesian coordinates) to be poorly represented.

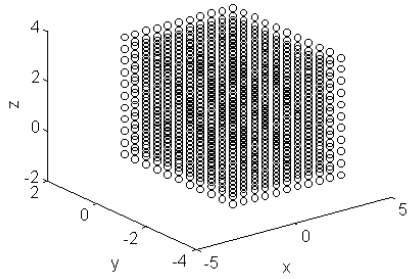

Figure 2. *Distribution of training data points in the joint space (inputs to the forward model, outputs for the inverse kinematics neural network)*

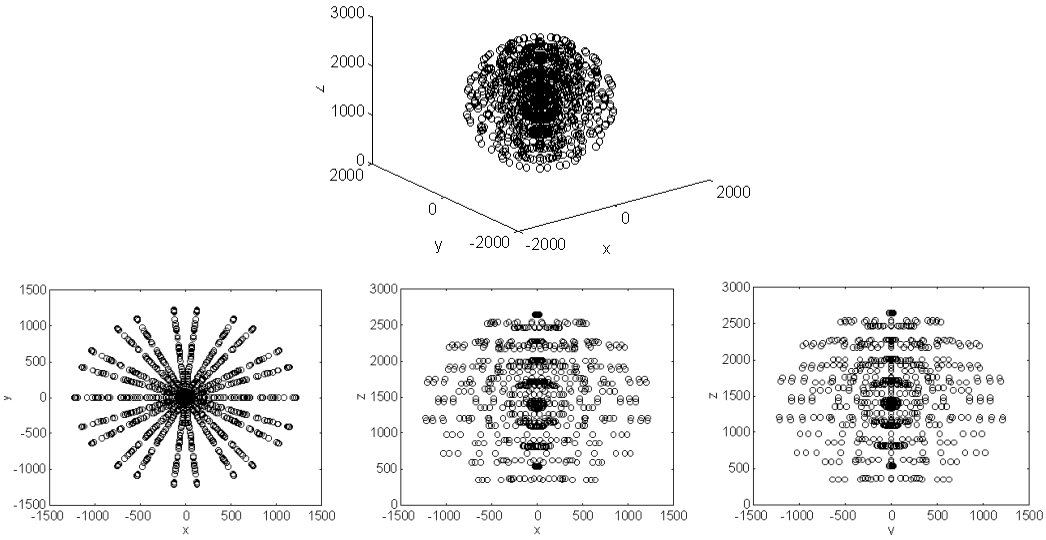

Figure 3. *Distribution of training data points in the Cartesian workspace (outputs of the forward model, inputs to the inverse kinematics neural network)*

This usually causes problems when used with feedforward backpropagation neural networks. Moreover there is no straight way to force the network to emphasize to specific portions of interest within the workspace where the manipulator is meant to operate and to suppress unrelated portion of the workspace that the robot may never need to reach.

A training set consisting of pairs that are not uniformly distributed in the input space, inhibits the use of some alternative network architectures. The well-known *radial basis function networks* (RBFNs) rely much of their performance in this uniformity since one neuron of their hidden layer is associated with each input pattern or a cluster of input patterns. The overall system output for a random input is formulated roughly as an interpolation between these memorized points. This makes the RBFNs quite sensitive on the training data distribution.

4.2. Multiple Solutions

Even for non-redundant manipulators, i.e. for manipulators whose number of joints is less or equal to the dimension of the workspace, the inverse kinematics problem has not always a single solution. This means that several (quite different) joint coordinates may lead the end-effector to the same (or very similar) Cartesian workspace coordinates. Thus, inverting the input-output pairs to feed them to the supervised learning algorithm, the network will be taught to respond in the same (or very similar) input with various (or very different) outputs.

Such data inconsistencies will, of course, either inhibit learning from converging or result to a network that produces a mix of all the associated outputs at the presence of an 'inconsistent' input.

So, making a system invertible is not always easy or even possible, and can only be done at the cost of adding extra inputs to the network (e.g. configuration information as 'left shoulder elbow above wrist') or designing a system of more complex architecture and/or learning algorithm.

4.3. Minimization of Misleading Error Criterion

The neural network will accept as inputs the workspace Cartesian coordinates and produce as output the joint coordinates. So, during supervised learning the error propagated will be the difference between the target joint values and the output of the network.

This is quite misleading since our basic aim is to design a system that will minimize the error between the workspace coordinate *command* and the workspace coordinates *actually attained* by the robot. In some circumstances a very small error in the joint coordinates may yield a very large error in the workspace coordinates. Equivalently we may say that the system trained in such a way, is essentially an *open* control system in the Cartesian space.

This way, even a fairly large multi-layer perceptron trained with plain backpropagation is not able to reduce the error to zero. This, of course, is not due to the network since when the closed form analytical equations of the system can be calculated, supervised learning of this network may indeed lead the error virtually to zero.

However, if we can manage to decrease this 'misleading' error criterion enough we can expect that the resulting network will perform accordingly well, i.e. the command and actually attained Cartesian coordinates will be as close as desirable.

Some additional problems of the application of 'plain' multi-layer perceptron neural networks in inverse modeling that do not have their source in the special features of the inverse kinematics problem but in the network itself, include the too large training periods required by backpropagation and the inefficiency of selecting the initial weights randomly.

4.4. Learning Algorithm

The generalized delta rule used by the plain backpropagation learning algorithm to compute the gradient needed for the steepest descent, achieves low learning rates in virtually every problem. Moreover, the accuracy obtained after the network converges (if it does at all), is often too poor for the network to be used to real-world robotic applications.

The performance of steepest descent methods in the vicinity of a minimum is very poor and consists one of the main reasons for its low overall performance.

4.5. Selection of Initial Weights

Random selection of initial weights may lead many of the neurons fast to saturation, thus having minor contribution to the overall network output. Updates of such neurons by the learning algorithm are useless. By selecting initial weights in such way, there exists no way to force sufficient number of neurons to the active region of the network in order for them to participate in the formulation of the desired mapping.

5. Neural Inverse Kinematics Problems and their Treatment

The aforementioned problems need to be treated in order to obtain a system that can effectively address the inverse kinematics problem using neural networks so that accuracy, robustness, and training time can be acceptable.

5.1. Generating the Training Patterns

A typical data producing scheme, would loop though the permissible values of the joint angles and calculate their respective workspace coordinates through the forward model to produce a set of forward kinematic solution pairs. Figures 2 and 3 contain such data as produced by the forward model of the 3R robot described above by looping for each of the joints with a constant step.

Ideally, this set of pairs if reversed would provide sufficient data to a supervised learning algorithm for inverse kinematics. But that is not the case for two main reasons:

- Multiple solution pairs are present in the data set, i.e. a specific point in the workspace may be reached by two or more different configurations of the robot, so two or more of the inverted pairs could have identical or very similar inputs (workspace coordinates) related to two very different outputs (joint coordinates). This would render the data inconsistent.
- The distribution of training pairs in the input universe after reversing inputs and outputs turns out to be not as uniform as the Fig. 2, but rather more like Fig. 3 seriously affecting the generalization or even the convergence itself of a neural network that would be trained upon them.

These two problems definitely need to be sorted out for an adequate training set to be obtained.

5.2. Special Architectures and/or Learning Algorithms for Multiple Solutions

Obviously, for the efficient control of a robot arm, *all* solutions should be available to the task planner when a path is generated for the end-effector in the Cartesian coordinate space.

Unfortunately, most researchers do not explicitly describe strategies to address the multiple solutions problem. Suggested methodologies are usually tested through simulation of a two-

or three-degrees-of-freedom planar robot limiting the Cartesian or joint coordinate space so as to exclude multiple solutions.

Examples of related work given by are Jordan [15] and Yabuta *et al.* [16] who suggested solutions for the inverse kinematics calculation by neural networks with a special scheme to deal with this problem.

A quite obvious and not too 'expensive' approach, uses one neural network module per configuration. So, in the case of positioning the 3R robot, we would need four neural network modules for inverse kinematics. This is quite reasonable since:

- Trying to assign multiple tasks to a neural network results in lengthy training periods and questionable convergence due to the 'spatial crosstalk' phenomenon. This arises when the network tries to learn two quite different mappings and mainly results from the distributed nature of the representation inside the network.
- It keeps the system more modular without requiring much more resources. A rather small network proves to be quite effective in dealing with each configuration alone.
- Such an architecture conforms with the '*mixture of local experts*' architecture as proposed by Jacobs *et al*. [18], where a set of neural networks appropriately coordinated by a so called 'gating network' are competing to learn different aspects of a problem.

Peeking at the kinematic properties of the robot at hand, one may deal with separating the data corresponding to different configurations by selectively limiting the ranges of the loops used to produce them. For example, one could restrict the 3R robot to the left and above configuration by imposing the conditions $\theta_2 > -90°$ and $\theta_3 > 90°$.

A different way to handle multiple configurations, is to apply the '*mixture of local experts*' architecture (also referred to as 'modular neural network' [20]) and let each of the various networks specialize to different configurations through competition. Of course, although this approach seems to be very attractive since it requires no prior information relatively to the manipulator except of the *number* of configurations, it involves various risks and certainly requires more training time since the gating network needs to figure out the different configurations by itself so as to assign one neural network per configuration. Moreover, it is not appropriate for the case of redundant robots where the number of configurations may be infinite.

5.3. Rendering the Data Distribution More Uniform

The distribution shown in Fig. 3 concerning the distribution of learning samples in workspace coordinates is far from acceptable. What we would like, is to have more uniformity for the following reasons:

- To facilitate the network's convergence during the learning phase.
- To improve its generalization capabilities during recall.
- To be able to efficiently make use of network architectures alternative to multi-layer perceptrons, like *radial basis function networks* by directly assigning the centers the Gaussian functions of their prototype layer to coincide with carefully selected 'representative' exemplar patterns.

Actually, it would be desirable that our input rather than our output data have the distribution of Fig. 2. Our advantage is that we have a forward kinematic module available. So, a large

number of training pairs can be produced off-line. Then, from these data we can explicitly select a set whose Cartesian coordinates are matching the desired distribution.

In the case where a closed form inverse kinematics solution is available, one can make use of it to appropriately choose uniformly (with respect to the Cartesian coordinate inputs) distributed data. This approach was *not* adopted so as to retain the generality and not to restrict the investigation to robots with closed form inverse kinematics solution. This choice is respected throughout the description.

For the exemplar 3R robot a large number of such data was produced. Then a lattice was defined on the Cartesian workspace as shown in Fig. 4. This lattice represents the desirable input data distribution.

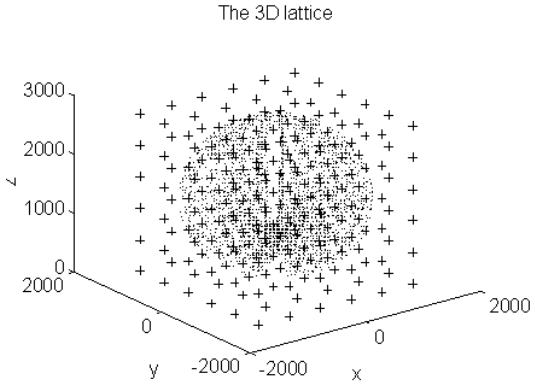

Figure 4. *A lattice defined on the Cartesian workspace of the robot*

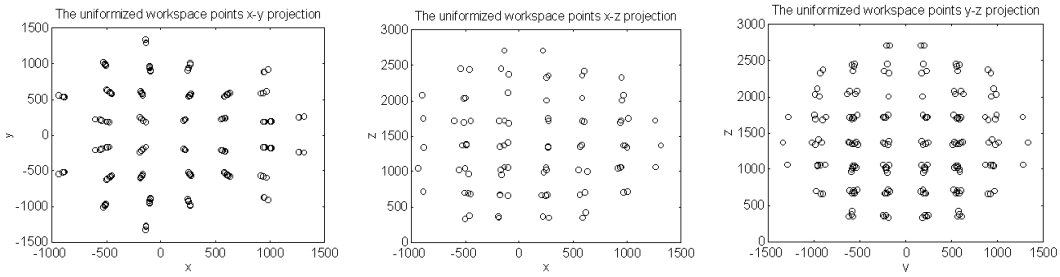

Figure 5. *A more uniform distribution of the patterns of Fig. 3*

Well known algorithms may be then used to isolate the samples that are close to the nodes of the lattice. These will become the training set, while the others will be used as the test set. Fig. 5 shows the result of such an algorithm for the case of the 3R robot.

It is obvious that the set of pairs collected with this method may match the desired distribution (within the workspace of the robot, of course) in an arbitrary degree depending solely on the number of pairs initially produced and the matching criterion imposed to classify a pattern as ‘belonging’ to a certain node. From a certain point and further, no important advantage is really gained by matching the desired distribution to higher degrees.

One of the special features of this approach is that the number of training patterns required is very limited comparing to the number of test patterns, minimizing thus the required training time to achieve a desired error level.

A different method for the self-generation of training patterns has been proposed by Albus for his CMAC [5] and Kuperstein for sensory-motor coordination [6]. According to this method, once the network has been sufficiently well ‘bootstrapped’ using randomly generated training

patterns, it can be used to produce a set of training patterns that approximately yield inputs in the correct distribution. More specifically, if we have a set of training inputs (forward model outputs), say O, that we would like to be present in the training set, we generate the training set using pairs:

$$\left(o_o(f(o_t)), f(o_t)\right),\ o_t \in O$$

where $o_o(\mathbf{x})$ is the observed system output vector when $\mathbf{x}$ is applied, and f denotes the 'bootstrapped' network's response. This way, network performance is improved by bringing the distribution of the next training patterns more in line and thus allowing for even better network performance. In [6], a 3% accuracy was achieved after about 1200 iterations. However, this method does not take advantage of a forward model when available and thus is more suited for the cases where training data are either 'expensive' or hard to obtain.

5.4. Data Scaling

Often, data need to be scaled so as to fit the requirements imposed by the selected neural network architecture and/or learning rule. In the case of networks with sigmoid or hard limiter transfer function for the output layer, it is obvious that the data should be in accordance. For practical applications, a table is created containing the minimum and maximum values for each input and output which is used to scale data before presenting them to the network.

5.5. Learning with an Emulator

Clearly, to address the problem of minimizing the wrong error function, a way is required to:

- apply the desired workspace coordinates, say p_d, to the network,
- obtain the joint coordinates, say q_a, from the network's output,
- apply this action to the forward model to obtain its true output, say p_a, and
- convert the 'true' system error, $p_d - p_a$ (and not $q_d - q_a$), back into joint errors in some way.

In bibliography [4, 7], it is common that a *differentiable* model of the *forward* system transfer equations —an 'emulator'— is used to perform this conversion elegantly. A simple implementation of this model uses a multi-layer perceptron. This model network can be trained using a 'plain' supervised learning scheme since the forward kinematics problem does not suffer from multiple solutions. Backpropagation can be used to derive all $\partial p/\partial q$ required with the difference that it will also be used for the input units of the emulator and not only for its hidden units. A diagram of such a system is shown in figure 6.

In [17] the learning algorithm ensures that the cost function used to train the network involves the error that is actually observed between the position command and the actual position attained by the robot. But on the other hand, the multiple solutions problem is bypassed by considering an appropriately constrained two-link planar manipulator.

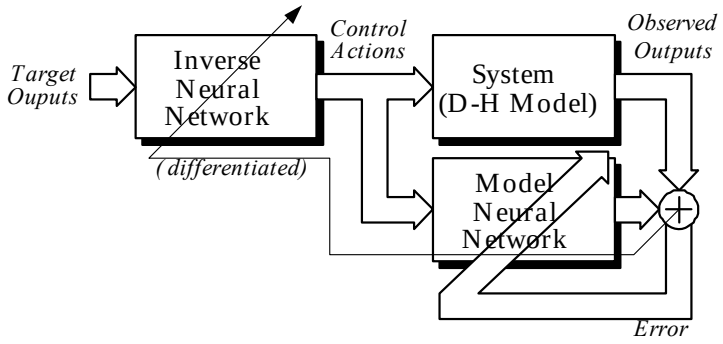

Figure 6. *Training a controller using a trained differentiable model of the system*

5.6. Selecting the Network Architecture and the Learning Algorithm

Many kinds of neural networks can be used to infer the mapping required by inverse kinematics. One of the most commonly used networks in robotics is the multi-layer perceptron equipped with the backpropagation algorithm while, depending on the specific application, an alternative may be provided by recurrent neural networks such as the Hopfield. Although radial basis function neural networks possess certain very attractive characteristics, their use in robotics and inverse kinematics specifically, is comparatively limited.

The inflexibility of 'plain' backpropagation, has partly put it aside especially in the presence of more recent and promising learning rules as the delta-bar-delta, quasi-Newton methods, the Levenberg-Marquardt algorithm etc., and more efficient (at least for some specific problem domains) network architectures as the radial basis function networks.

Research on methods to accelerate the backpropagation learning algorithm fall into two major categories [9]:

- *ad-hoc techniques*: These include ideas as using momentum terms and rescaling variables, varying the learning rate, etc.
- *standard numerical optimization techniques*: The most popular of these used conjugate gradient or quasi-Newton methods [8, 10]. Newton's method can successfully *complement* steepest descent methods but by the cost of increased memory and resource requirements since the Hessian needs to be evaluated and inverted, and by the non-triviality of optimally switching between steepest descent and Newton's method. An even more efficient technique that is based on the Marquardt algorithm for nonlinear least squares is presented in [9].

Due to the computational load and memory requirements, the standard numerical optimization methods are restricted to multi-layer perceptron networks consisting of no more than a few hundred weights. When this is the case, these methods can offer an improvement of about two orders of magnitude with respect to the rate of convergence compared to the steepest descent technique.

5.7. Radial Basis Function Networks

A radial basis function network (RBFN) contains a hidden (prototype) layer of radially symmetric and bounded transfer function in its hidden layer. A thorough investigation of RBFNs can be found in [20].

RBFNs can be used virtually to any problem where a backpropagation network would be considered providing advantages like much faster training, formation of better decision boundaries, etc. Its main disadvantages, though, are usually the larger number of hidden nodes required since backpropagation networks give a more compact distributed representation.

Training a RBFN consists of two phases:

- A *clustering phase*, when input data are formed into clusters, using for example a k-means algorithm, and updating the weight from the input to the prototype layer so that each function is centered at each cluster and its radius is appropriately adjusted.
- An actual *learning phase*, where the weights from the prototype to the output layer are trained using an error learning rule.

By the uniform input data distribution that can be achieved using the lattice scheme described above, the clustering phase becomes quite safe and fast. Ideally, we could assign one prototype node (i.e. one radial function) for each of the lattice nodes. In practice, less nodes were found to be needed for the specific application.

5.8. Geometric Interpretation of Neuron's Functional Behavior for the Selection of Initial Weights

It is argued in [11], that by systematically selecting the initial values of the adaptive weights, the learning time may be reduced.

In [12], a systematic way for choosing the initial weight values is described that is based on a geometric interpretation of the neural network's functional behavior. The network used is a traditional backpropagation neural network with sigmoid transfer functions. The authors claim that this method considerably reduced the learning time and even caused the network to converge in many cases where the random selection of initial weight failed.

5.9. Building Prior Information into the Network Design

Multi-layer perceptrons are proven to be 'universal approximators', that is, they are able of approximating *any* function to an arbitrary accuracy. This is their strength but might as well be their weakness.

When addressing a problem category, more efficient solution strategies could be derived by making use of all the available a priori knowledge related to the category as long as the generality of the approach is sufficiently preserved.

One of the four commonsense rules suggested by Anderson [19] states that: "*Prior information and invariances should be built into the design of a neural network, thereby simplifying the network design by not having to learn them*". This is the idea behind the hierarchical neural networks as proposed by Guez and Selinsky [21]. Observing that some nonlinear functions were central to the control of robot dynamics, they trained them into some multi-layer perceptrons off-line and then used them as inputs to a final layer that learned to appropriately combine them so as to produce the overall solution.

Most open-chain industrial manipulators, may be broken down to two kinematic sub-chains, the first being primarily responsible for positioning the end-effector while the second for orientating it. Thus, the inverse kinematic problem for these manipulators may be broken down to two simpler ones, namely calculating the joints coordinates for the first sub-chain based on the desired Cartesian position of the end-effector, and calculating the joints

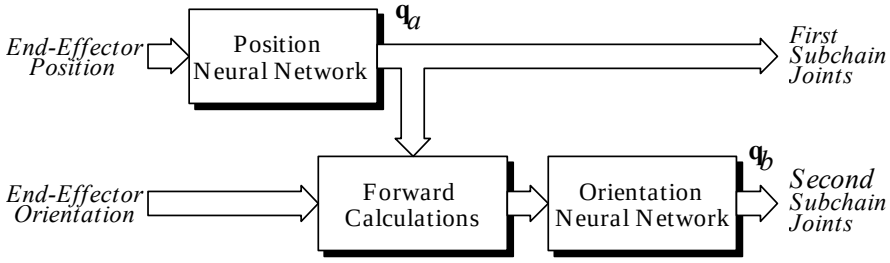

Figure 7. *Connecting the two networks*

coordinates for the second sub-chain based on the desired orientation of the end-effector and the joint coordinate values obtained for the first sub-chain.

Moreover, the kinematic articulation of the second sub-chain usually coincides with the definition of Euler angles, so in some cases the second part could consist of just a few straightforward geometric calculations instead of a neural network.

For a typical industrial manipulator consisting of 6 degrees of freedom (joints), this means that a 6-by-6 problem is broken down to two 3-by-3 problems. It is obvious that the complexity of the former is considerably higher than the complexity of the latter.

The 3R non-planar robot that served as an example up to now, is actually the basis for many of the common industrial robots. It is kinematically identical to the first three DOFs of the PUMA, the CLOOS and other commonly used manipulators. Moreover, the solution for their last 3 DOFs is trivial since it reduces to finding a set of Euler angles from an 3×3 orientation matrix. So, since the decomposition described above is applicable to these robots, this discussion is also valid for them too. Restricting our focus to the first three DOFs, the multiple joint solutions that exist in the general case for an arbitrary positioning of the robot reduce to four, namely left/right shoulder and above/below arm.

It is quite easy to notice that different joint variables depend on different Cartesian coordinates. Namely, for the left and above configuration, θ_1 depends solely on x and y (not on z) and θ_2, θ_3 seem to depend on all x, y, and z. Actually, after a closer look, θ_2 and θ_3 really depend on the distance of the end-effector from the z axis of the world coordinate system and not by x and y explicitly. That is, they depend on z and $r = \sqrt{x^2 + y^2}$ (i.e. two instead of three inputs).

So, instead of letting the network figure out this kind of dependency, we may hardwire this in by supplying r as an additional input. A similar approach was used by Schöneburg *et al*. [22] where except the Cartesian coordinates of the target end-effector position they also provided the network with additional inputs as the sine and cosine of the first and second axes angles.

Undertaking the approach adopted in [12] that uses one neural network per joint so as to eliminate 'spatial crosstalk', we may end up with three small neural networks with input-output spaces of 2×1 each. Thus a major decomposition has taken place solely based on observations of the kinematic structure of the robot at hand. Similar simplifying observations can be made for most manipulators.

Training such small neural networks is quite trivial while training 3×3 is certainly more demanding especially when more than one hidden layers are required, which is the usual case.

5.10. Pruning the Network

Pruning is a concept introduced by Rumelhart that attempts to minimize both network complexity and square error over a training set. Pruning provides the means for reducing the 'overfitting' phenomenon since a network of minimal complexity that performs well on a training set is expected to generalize better than a more complex one. The simplest way of minimizing a network is by removing the relatively small weights.

6. Fuzzy and Neurofuzzy Approach to Inverse Kinematics

The application of the fuzzy methodology will be shown by considering an articulated 3-DOF robotic manipulator (Fig. 1). In this case Eq. (1) is specialized as:

$$p_x = a_2 c_1 c_2 + a_3 c_1 c_{23} \tag{5a}$$

$$p_y = a_2 s_1 c_2 + a_3 s_1 c_{23} \tag{5b}$$

$$p_z = a_1 + a_2 s_2 + a_3 s_{23} \tag{5c}$$

where

$$c_i = \cos q_i, \quad s_i = \sin q_i \quad (\mathrm{i} = 1,2), \quad c_{23} = \cos(q_1 + q_2), \quad \mathbf{p} = [p_x, p_y, p_z]^T$$

in the position of the robot tip in the Cartesian space; a_1, a_2, a_3 are the lengths of the links of the robot, and q_i (i = 1,2,3) are the angles of the links as shown in Fig. 1. The analytical solution of the inverse kinematics problem is given by [38, 39]:

$$q_1 = a\tan 2[\frac{p_x}{p_y}] \text{ or } q_1 = a\tan 2[\frac{-p_x}{-p_y}] \tag{6a}$$

$$q_2 = a\tan 2[\frac{(p_z - a_1)(a_2 + a_3 c_3) \mp a_3\sqrt{{p_x}^2 + {p_y}^2}}{(p_z - a_1)a_3 s_3 \pm (a_2 + a_3 c_3)\sqrt{{p_x}^2 + {p_y}^2}}] \tag{6b}$$

$$q_3 = a\tan 2[\frac{\pm\sqrt{4{a_2}^2{a_3}^2 - [(p_z - a_1)^2 + {p_x}^2 + {p_y}^2 - {a_2}^2 - {a_3}^2]}}{(p_z - a_1)^2 + {p_x}^2 + {p_y}^2 - {a_2}^2 - {a_3}^2}] \tag{6c}$$

which reveal the existence of multiple solutions for q_i (i=1,2,3).

The linearized version of (6a-c) about some nominal configuration has the following matrix form:

$$\delta_{\mathbf{p}} = [P_{ij}]\delta_{\mathbf{q}}, \quad \mathbf{p} = \begin{bmatrix} \mathrm{p_x} \\ \mathrm{p_y} \\ \mathrm{p_z} \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} \mathrm{q_x} \\ \mathrm{q_y} \\ \mathrm{q_z} \end{bmatrix} \tag{7}$$

where $\delta_{\mathbf{q}}$ are a small variations $(\delta_{q_i} << 1)$, and

$$p_{11} = \frac{\partial\ \mathrm{p_x}}{\partial\ \mathrm{q_1}} = -(a_2 s_1 c_2 + a_3 s_1 c_{23}) \tag{8a}$$

$$p_{12} = \frac{\partial\ \mathrm{p_x}}{\partial\ \mathrm{q_2}} = -(a_2 s_2 c_1 + a_3 c_1 s_{23}) \tag{8b}$$

$$p_{13} = \frac{\partial\ \mathrm{p_x}}{\partial\ \mathrm{q_3}} = -a_3 c_1 s_{23} \tag{8c}$$

$$p_{21} = \frac{\partial\ \mathrm{p_y}}{\partial\ \mathrm{q_1}} = a_2 c_1 c_2 + a_3 c_1 c_{23} \tag{8d}$$

$$p_{22} = \frac{\partial\ \mathrm{p_y}}{\partial\ \mathrm{q_2}} = -[a_2 s_1 s_2 + a_3 s_1 s_{23}] \tag{8e}$$

$$p_{23} = \frac{\partial\ \mathrm{p_y}}{\partial\ \mathrm{q_3}} = -a_3 s_1 s_{23} \tag{8f}$$

$$p_{31} = \frac{\partial\ \mathrm{p_z}}{\partial\ \mathrm{q_1}} = 0 \tag{8g}$$

$$p_{32} = \frac{\partial\ \mathrm{p_z}}{\partial\ \mathrm{q_2}} = a_2 c_2 + a_3 c_{23} \tag{8h}$$

$$p_{33} = \frac{\partial\ \mathrm{p_z}}{\partial\ \mathrm{q_3}} = a_3 c_{23} \tag{8i}$$

Following the results of [34-37] all variables $\delta p_x, \delta p_y, \delta p_z, \delta q_1, \delta q_2$ and δq_3 are fuzzified using the following seven fuzzy sets(although for higher accuracy more fuzzy sets might be used):

PL	Positive Large
PM	Positive Medium
PS	Positive Small
Z	Zero
NS	Negative Small
NM	Negative Medium
NL	Negative Large

Since Eq. (7) is a linear model, the superposition principle is applicable for the fuzzy associative memory (FAM) of the robot kinematics.

In [37] the following rules were used to determine δq_1 for a given $\delta p_x, \delta p_y$ and δp_z.

Table 1: Fuzzy rules for the linearized kinematics

	δp_x						
δq_1	NL	NM	NS	Z	PS	PM	PL
NL	PL	PM	PS	Z	NS	NM	NL
NM	PL	PM	PS	Z	NS	NM	NL

p_{1i}	NS	PL	PM	PS	Z	NS	NM	NL
	Z	Z	Z	Z	Z	Z	Z	Z
	PS	NL	NM	NS	Z	PS	PM	PL
	PM	NL	NM	NS	Z	PS	PM	PL
	PL	NL	NM	NS	Z	PS	PM	PL

The entries of the above fuzzy table (FAM) can be obtained via inspection of a three-dimensional graph. Similar FAMs have to be constructed for p_{2i} and p_{3i} (i=1,2,3) in order to determine δq_1. Each entry of the FAM corresponds to a rule of the type:

IF (p_{1i} is NM) AND (δp_x is PS) THEN δq_1 is NS

The overall FAM (fuzzy look-up table) consists of 49 rules, and therefore for p_{1i}, p_{2i} and p_{3i} one needs a total of 147 rules for determining the inverse kinematics solution, which implies a very high computational load. Therefore we must look for methods of reducing the computational effort. The above technique, although theoretically correct, is not suitable for automatically solving the inverse kinematics problem, since it needs the construction of a 3-D graphical representation of the FAM which is a difficult job.

An alternative method consists in collecting a set of training samples from measurements of q_i (i=1,2,3) obtained by moving the robot to a set of different positions p_x, p_y and p_z (Fig. 8).

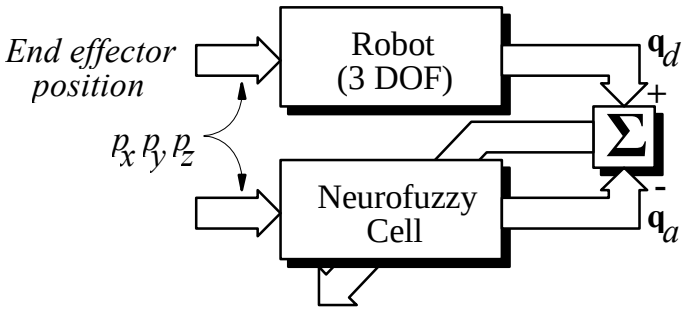

Figure 8. *Neurofuzzy structure for robot inverse kinematics;* $\mathbf{q}_d$: *desired joint angle vector,* $\mathbf{q}_a$: *joint angle vector actually provided by the neurofuzzy cell*

Then, a neurofuzzy network can be trained with a sufficient number of data points [40-42]. Here a number of issues similar to those studied in sections 4 and 5 may be considered. In particular, the membership functions and the network weights have to be adjusted so as to obtain the desired minimum error. In practice, instead of moving the actual robot to different positions for generating the required training data, one can use the forward kinematic equations which can be easily and uniquely computed. This has the additional advantage of selecting the desired angles at the design phase and producing a unique mapping between the angles and the corresponding Cartesian positions.

7. Simulation Results

Standard matrix operations may be used to calculate the forward kinematics of the any robot using the D-H methodology. Assuming that the kinematic parameters of the robot are *accurate* we may relay on it to produce valid workspace coordinate and joint coordinate pairs.

Data has been pre-processed following the techniques as described in the previous sections (figures 2, 3, 4, and 5 depict the various steps of the actual simulation).

RBFNs were used as the underlying neural network model. Our choice was to start with relatively large networks and to prune, after learning had converged, the nodes that less participated in the output, i.e. the prototype nodes whose weight received small values comparing to the values of the others.

The approach undertaken to face the spatial crosstalk and the multiple solutions as described above, led to the overall system architecture illustrated in Fig. 9.

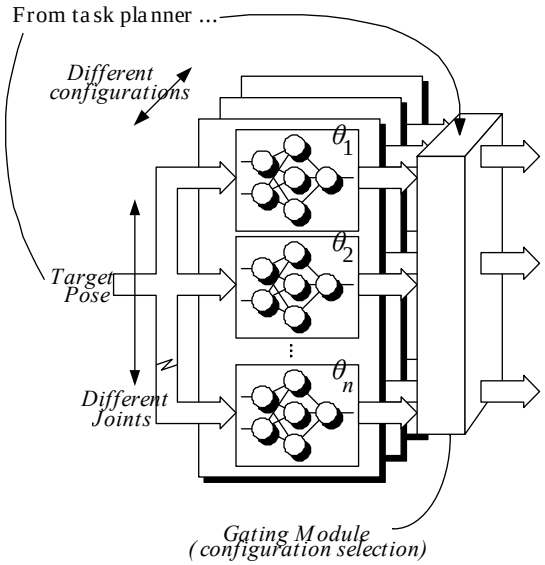

Figure 9. *The overall system architecture*

Due to the efficiency of RBFNs and the adequate data pre-processing, the networks converged quite fast. Training was stopped for each network when error started to decrease very slowly. Pruning took place by periodically removing the weights whose magnitudes were below a threshold percent of the maximum of the absolute magnitude of all the weights appearing in the network. In some cases, the results of pruning were remarkable.

The final system error managed to drop under 2%. By using more data produced by the forward model, a more dense lattice during preprocessing, and more hidden neurons at the prototype layer the error may be further decreased.

The corresponding results obtained by the neurofuzzy system proposed in [42], with the input-output variables quantized in five intervals and an MLP with one hidden layer, showed an RMS error less than 2%.

8. Conclusions

A general architecture along with related techniques to face the difficulties of the application of neural networks to the inverse robot kinematics problem were presented. An algorithm was described that may be used to improve the quality of the training pairs and to make possible and much more efficient the use of neural networks alternative to the multilayered perceptrons and to training algorithms alternative to the backpropagation of error. Simulation results were presented for the case of a 3R non-planar robot arm with a very common

kinematic structure. Throughout the discussion the robot was assumed to be calibrated. If this is not so, the same architecture still proves to be very efficient.

Two fuzzy logic methods were examined via a FAM and a neurofuzzy structure. It was assumed that knowledge of the forward kinematics of the robot at hand was available, through, for example, the D-H kinematic parameters which were assumed to be known and *accurate*. However, no knowledge was assumed relatively to the inverse kinematics of the robot at hand. So this approach is quite generic and thus applicable to a wide range of robotic manipulators. If the D-H model is used to obtain forward kinematics solution pairs then it is implicitly assumed that the robot is calibrated. If that is not the case the actual robot along with a positioning sensor will have to be used to produce the required data.

References

[1] G. Wu and J. Wang, "A Recurrent Neural Network for Manipulator Inverse Kinematics Computation," *Proc. IEEE Intl. Conf. on Neural Networks (ICNN'94)*, Florida (1994)

[2] J. Guo and V. Cherkassky, "A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing," *Proc. IEEE Intl. Conf. on Neural Networks*, San Diego, California, pp. 617-621 (1988)

[3] Y. Li and N. Zeng, "A Neural Network Based Inverse Kinematics Solution In Robotics," *Neural Networks in Robotics*, G. A. Bekey and K. Y. Goldberg, eds., Kluwer, pp. 97-111 (1993)

[4] D. Psaltis, A. Sideris, and A. Yamamura, "A Multilayered Neural Network Controller," *Proc. IEEE Intl. Conf. on Neural Networks*, San Diego, California, pp. 17-21 (1987)

[5] J. S. Albus, "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Transaction of the ASME*, pp. 220-227, Sept. (1975)

[6] M. Kuperstein, "Implementation of an Adaptive Neural Controller for Sensory-Motor Coordination," *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman-Soulié, and L. Steels (Eds.), *Elsevier*, North-Holland

[7] A. Cousein, "Neural Networks for Robot Control," *Intl. Conf. on Software Engineering for Real-Time Systems*, Cirencester, UK, pp. 119-124 (1991)

[8] H. S. M. Beigi and C. J. Li, "A New Set of Learning Algorithms for Neural Networks," *Proc. ISMM Conf. on Computer Architectures in Design, Simulation, and Analysis*, New Orleans, Louisiana (1990)

[9] M. T. Hagan and M. B. Manhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Proc. Neural Networks* (1994)

[10] R. Battiti, "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method," *Neural Computation*, Vol. 139, No. 3, pp. 301-310 (1992)

[11] D. Nguyen and B. Widrow, "Improving the Learning Speed by Choosing Initial Values of the Adaptive Weights," *Proc. IJCNN*, San Diego, pp. 111-121 (1990)

[12] M. Shoham, C. J. Li, Y. Hachman, and E. Kreindler, "Neural Network Control of Robot Arms," *Annals of the CIRP*, Vol. 41, No. 1, Technion, Israel Institute of Technology, Haifa, Israel, pp. 407-410 (1990)

[13] A. Guez and Z. Ahmad, “Solution to the Inverse Kinematics Problem in Robotics by Neural Networks,” *IEEE Intl. Conf. on Neural Networks (ICNN)*, San Diego, California pp. 617-621 (1988)

[14] A. G. Barto, “Connectionist Learning for Control,” *in Neural Networks for Control*, MIT Press, Cambridge, Massachusetts, pp. 5-58 (1990)

[15] M. I. Jordan, “Supervised Learning and Systems with Excess Degrees of Freedom,” *COINS Technical Report*, pp. 88-27 (1988)

[16] T. Yabuta, T. Tsujimura, T. Yamada, and T. Yasuno, “On the Characteristics of the Robot Manipulator Controller Using Neural Networks,” *IEEE Intl. Workshop on Industrial Applications of Machine Intelligence and Vision*, MIV-89 (1988)

[17] M. M. Gupta and D. H. Rao, “General Learning for Robot Coordinate Transformations Using Dynamic Neural Network,” *Proc. SPIE Conf. on Intelligent Robots and Computer Vision*, vol. SPIE-2055, pp. 524-535 (1993)

[18] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive Mixtures of Local Experts,” *Neural Computation*, 3, pp. 79-87 (1991)

[19] J. A. Anderson, “General Introduction,” In *Neurocomputing: Foundations of Research* (J. A. Anderson and E. Rosenfeld, eds.), pp. xiii-xxi, Cambridge, MA: MIT Press (1988)

[20] S. Haykin, “Neural Networks,” *Macmillan*, NY (1994)

[21] A. Guez and J. Selinsky, “Neurocontroller Design via Supervised and Unsupervised Learning,” *J. Intell. and Robotic Systems*, Vol. 2, pp307-335 (1989)

[22] E. Schöneburg, N. Hansen, and A. Gawelczyk, “Neuronale Netzwerke, Einführung, Überblick und Anwendungsmöglichkeiten,” Msrkt-u, *Technik-Verlag* (1990)

[23] R. P. Paul, “Kinematic Control Equations for Manipulators,” *IEEE Trans. on Systems, Man, and Cybernetics* (1981)

[24] X. Zhang, J. Lewis, F. L. N-Nagy, “Inverse Robot Calibration Using Artificial Neural Networks,” *Engng. Applic. Artif. Intell.*, Vol. 9, No. 1, pp. 83-93 (1996)

[25] Y. T. Li and K. Han, “A Lyapunov Based Cartesian Trajectory Control for Robot Manipulators,” *Chinese J. of Automation*, Allerton Press, Vol. 17, No. 1 (1991)

[26] Y. T. Li and K. Han, “A Variable Structure Control Scheme for Robot Cartesian Tracking,” *Chinese J. of Automation*, Allerton Press, Vol. 18, No. 3 (1992)

[27] D. E. Whitney, “Resolved Motion Rate of Manipulators and Human Prostheses,” *IEEE Trans. Man Machine and Systems*, MMS-10, pp. 47-53 (1969)

[28] P. Gupta and N. K. Sinha, “Control of Robotic Manipulators Using Neural Networks: A Survey,” In: S. G. Tzafestas, ed., *Intelligent Control Methods and Applications*, Kluwer, Dordrecht/Boston, pp. 103-136 (1997)

[29] B. F. J. Artaga, “Multilayer Back-Propagation Network for Learning the Forward and Inverse Kinematic Equations,” *IEEE Intl. Joint . Conf. On Neural Networks (IJCNN ’90)*, Washington, D.C., Vol.2, pp. 319-322 (1990)

[30] A. Guez and Z. Ahmad, “Accelerated Convergence in the Inverse Kinematics via Multilayer Feedforward Networks,” *Proc. IEEE Intl. Joint Conf. on Neural Networks (IJCNN ’89)*, Washington, DC, vol.2, pp. 341-344 (1989)

[31] L. Nguyen, R. Patel and K. Khorasani, “Neural Network Architectures for the Forward Kinematic Problem in Robotics,” *Proc. IEEE Intl. Joint Conf. on Neural Networks (IJCNN ’90),* pp. 393-399 (1990)

[32] S. G. Tzafestas, “Neural Networks in Robot Control,” In : S. G. Tzafestas and H. B. Verbruggen, *Artificial Intelligence in Industrial Decision Making, Control and Automation*, Kluwer, Dordrecht/Boston, pp. 327-387 (1995)

[33] J. Martinez, J. Bowles, and P. Mills, “A Fuzzy Logic Positioning System for an Articulated Robot Arm,” *IEEE Intl. Conf. on Fuzzy Systems* (1996)

[34] A. Nedungadi, “Application of Fuzzy Logic to Solve the Robot Inverse Kinematic Problem,” *Proc. 4th SME World Conf. on Robotic Research* (1991)

[35] A. Zilouchian, F. Hamono, and T. Jordanides, “Intelligent Control Using Artificial Neural Networks and Fuzzy Logic: Recent Trends and Industrial Applications,” In: S. G. Tzafestas (ed.), *Methods and Applications of Intelligent Control,* Kluwer, Dordrecht/Boston, pp. 69-102 (1997)

[36] S. G. Tzafestas and C. S. Tzafestas, “Fuzzy and Neural Intelligent Control: Basic Principles and Architectures,” In: S. G. Tzafestas (ed.), *Methods and Applications of Intelligent Control,* Kluwer, Dordrecht/Boston, pp. 25-67 (1997)

[37] D. W. Howard and A. Zilouchian, “Application of Fuzzy Logic for the Solution of Inverse Kinematics and Hierarchical Controls of Robotic Manipulators,” *J. Intelligent and Robotic Systems* (1998)

[38] R. P. Paul, Robot Manipulators: Mathematics, Programming and Control, *The MIT Press*, Cambridge, MA (1981)

[39] F. S. Fu, R. C. Gonzalez, and C. S. G. Lee, Robotics: Control, Sensing Vision and Intelligence, *McGraw-Hill*, New-York (1987)

[40] J. S. Jang, “Self -Learning Fuzzy Controllers Based on Temporal Back Propagation,” *IEEE. Trans. On Neural Networks*, Vol. 3, No. 5 (1992)

[41] J. S. Jang and C. Sun, “Neuro-Fuzzy Modelling and Control,” *Proc. IEEE*, Vol. 83, No. 3, pp. 378-406 (1995)

[42] S. G. Tzafestas, S. Raptis, and G. Stamou, “A Flexible Neurofuzzy Cell Structure for General Fuzzy Inference,” *Mathematics & Computers in Simulation*, Vol. 41, Nos. 3-4, pp. 201-208 (1996)