Adaptive Robot Control Using Artificial Neural Networks : An Application in the Theory of Cognition

by:

Ali M.S. Zalzala and Alan S. Morris

Robotics Research Group,
Department of Control Engineering,
University of Sheffield,
Mappin Street,
Sheffield S1 3JD,
U.K.

Research Report # 374

October 1989

RR # 374

Abstract

During the last decade the problem of real-time robot control has proven to be of extreme difficulty. At present, available control systems are inadequate for the task. In addition, the application of sophisticated control schemes such as the Model-Reference Adaptive Control is prevented by the heavy computational task that is necessary to implement it. This paper offers a feasable solution for the fast real-time control of robot manipulators by adapting certain concepts of neural networks. An adaptive controller is presented which solves for the highly coupled dynamic equations of motion, which are known to present the heaviest obstacle in real-time computations. A symbolic representation of the Lagrange-Euler equations is adapted for this purpose. The neural controller is designed on a multi-layered network, in which the adaptation for environment changes could be accommodated via the back-propagation of errors throughout the network. The proposed controller is shown to be capable of adapting to changes in the robot model parameters in real-time applications. In addition, the use of distributed parallel processing concepts greatly simplifies the computations required within each control cycle. Simulation results are reported for the full model of the Unimation PUMA 560 manipulator with 6 degrees-of-freedom, along with a performance analysis of an implementation of the proposed controller on a T800 transputer network.

1. Introduction

The control of robot manipulators has been considered a difficult task for a long time. Presently available control schemes use linear feedback measures to track the desired motion, thus ignoring the coupling between the manipulator joints, and the non-linear nature of the system [Craig 1986]. A slightly more sophisticated method includes a fixed model of the robot arm. However, such an approach does not take into account any changes in the payload during the execution of motion, in addition to the presence of uncertainties in the originally used model parameters. Therefore, in real-time application, ignoring all or parts of the robot nonlinear dynamics, or even errors in the parameters of the manipulator, may cause serious deviations and thus render such control strategies to be inefficient.

The essential drawback in a PID controller is the constant gain parameters in its linear feed-back compensation loop. Hence, although a complicated control scheme such as the Computed Torque Technique [Fu,Gonzalez and Lee 1987] could be tolerated through its implementation on fast state-of-the-art computing structures [Zalzala and Morris 1989a], such constant gains could produce substantial tracking deviations. One solution has been proposed by the introduction of different schemes for *adaptive robot control* [Dubowsky and DesForges 1979, Koivo and Guo 1983, Lee and Chung 1984]. Nevertheless, the proposed theory has been considered to present too heavy a mathematical burden for execution within the short control cycle usually required for fast trajectory tracking. Therefore, no efficient practical implementation has been reported for such schemes.

In recent years, the principles of distributed parallel processing offered by the concept of Artificial Neural Networks (*NN*) have been seen as a solution for the long-standing real-time robot control problem. Multi-layered neural nets have been employed in different aspects of robot control [Daunicht 1989]. Preliminary control structures were presented by several researchers, which defined a general framework [Guez,Eilbert and Kam 1988, Psaltis,Sideris and Yamamura 1987, Psaltis,Sideris and Yamamura 1988, Josin,Charney and White 1988, Elsley 1987], for which sensori-motor coordination was the main principle [Ritter,Martinetz and Schulten 1989]. In the kinematics control of a robot arm, a feed-forward layered network was proposed for the motion tracking of a two d.o.f. arm [Pao and Sobajic 1987] and later implemented on an actual robot manipulator [Sobajic,Lu and Pao 1988]. In addition a scheme has been proposed for the solution of the inverse kinematics algorithm for a three d.o.f. planar manipulator [Guez and Ahmad 1988]. Furthermore, artificial neural nets were proposed for the representation of navigation maps for autonomous path planning [Jorgensen 1987]. Adaptive dynamic control was tackled by several researchers as well. A preliminary analysis of the problem has been assessed [Guez,Eilbert and Kam 1987], while different approaches to the solution could be found [Bassi and Bekey 1989, Kawato,Uno and Isobe 1987, Kawato,Uno,Isobe and Suzuki 1988, Ritter,Martinetz and Schulten 1989].

In this paper, a new computational model is presented for the adaptive control of robot manipulators. Certain concepts of the neural network approach have been extracted to provide for a fast real-time controller. The neural controller is designed on a multi-layered network, for which the adaptation for the environmental changes could be accommodated via the back-propagation of errors throughout different layers. A symbolic representation of the dynamic equations of motion is considered, for which the proposed controller is shown to be adequate to accommodate any changes in the robot model in real-time applications. In addition, the use of distributed parallel processing concepts greatly simplifies the computations required within each control cycle. Simulation results are reported for the full model of the Unimation PUMA 560 manipulator with 6 degrees-of-freedom. In addition, an implementation of the neural network has been suggested, using the INMOS T800 transputer as the main block of the multiprocessor system.

The work in this paper is presented as follows; In section 2, the basic theory of cognition and neural networks is presented, while a statement of the problem is made in section 3. The symbolic representation of the dynamic equations is discussed in section 4, for which the distributed formulation is illustrated in section 5. The robot neural controller is designed in section 6, while its mathematical justification is included in section 7. Simulation results of a case study are shown in section 8. In addition, the practical VLSI implementation is included in section 9. Finaly, conclusions are drawn on the proposed neural controller.

2. The Theory of Cognition

2.1. Underlying Concepts

As far as engineering applications are concerned, combining distributed parallel processing of tasks along with the ability of learning is a distinctly new approach. The process of comparing engineering problems to the actual behaviour of living beings provides a unique step towards effective artificial intelligence systems. This emerging field is known as *connectionism*, or *artificial neural networks*. It obtains its terminology from the *neuron* cell which is an essential element in the living nervous system. The main aim of current research is to recast the problem in a suitable mathematical form, and further set up a model simulating the nervous system [Arbib 1987]. The connectionist network formed of different so-called neuron-like nodes would become the physical media for the required system, where all communications amongst nodes are maintained through passing electrical impulses. The authoritive text in this field is acknowledged to be that of the Parallel Distributing Processing Group (PDP) in the USA [Rumelhart and McClelland 1986], where different relevent models were established [McClelland and Rumelhart 1988]. A further good review of the PDP perspective can be found [Aleksander 1989].

2.2. Artificial Neural Nets

2.2.1. Main Structure

The main block in a neural network is the *neuron* node. As shown in figure (1), the node is a multiple-input single-output device with the following parameters :

- *Inputs to the node* : denoted as I_j , j=1,2,...,n, which could be external inputs or alternatively outputs of other neurons.
- *Weight of each input* : denoted as W_{ij}, linking the neuron i with others j=1,2,...,n. These are adapted during use in order to improve performance.
- *Combining function* (F^i_c) : Relates all inputs to the node, and can be either linear or nonlinear.
- *Activation function* (F^i_a) : the output of the neuron, which could be connected to any other node in the network, or even connected as a feed-back to the same node.

A one-to-one correspondance of this structure with its physical origin in brain cells can be established [Karna and Breen 1989]. However, it is very interesting to note that although modern high-speed VLSI structures can manipulate data much faster than the human brain, their capabilities as far as learning is concerned is quite restricted [Miche 1986]. This may be due to the brain operating in a highly parallel fashion.

A neural network is usually constructed of an input layer of neurons, an output layer and several intermediate, or *hidden*, layers. The number of hidden layers used along with the number of neurons in each of them depends on the learning requirements. In addition, a network may be either fully or sparsely connected. If fully connected, the output of each node in a layer is connected as an input to all nodes in the next layer. Alternatively, a network could be sparsely connected, where an example of a 4-layer network is shown in figure (2).

2.2.2. The Back-propagation Algorithm

Although several different schemes are available for the construction of multilayered neural nets [Lippmann 1987, Roth 1988], The *back–propagation* algorithm is the most common [Rumelhart,Hinton and Williams 1986a]. The back-propagation learning rule is based mainly on a combination of the least-mean-square and the gradient descent methods. It involves two phases in its operation, as follows :

1. *Forward phase* : includes inputing the desired values, computing throughout the network's layers and producing its outputs.
2. *Backward phase* : where the actual outputs of the network are compared to the desired values, and the errors resulting are propagated back through each of the layers, adjusting the weights of each node.

This process is continued for other inputs until the network *learns* how to behave in the desired manner. The back-propagation algorithm is illustrated in table (1), while its justification in mathematical terms can be found in the literature [Angus 1989].

Table (1) : The Back-Propagation Algorithm

```
I. Initialisation :
1. For each node in the network
    1.1. Initialise weights ($W_{ij}$) to small random values
    1.2. Initialise node biases ($\beta_i$) to small random values

II. Forward Phase :
1. Get inputs to the network
2. For each node $i$ in layer $l$
    2.1. Compute the combining function $F_c^i(I_j, W_{ij})$
    2.2. Compute the activation function $F_a^i(F_c^i, \beta_i)$

III. Backward Phase :
1. For each node in the output layer
    1.1. Calculate the error between the desired and actual values ($d_i - y_i$)
    1.2. Compute the error function, $\delta$, as
            $\delta_i = y_i(1-y_i)(d_i-y_i)$ †
2. For each node in other hidden layers
    2.1. Calculate the $\delta$ function as
            $\delta_i = F_a^i(1-F_a^i) \sum_j (\delta_j W_{ij})$ †
3. Update all weights in the network by
            $W_{ij}^{t+1} = W_{ij}^t + \eta \delta_j F_a^i + \alpha(W_{ij}^t - W_{ij}^{t-1})$
            where, $0 < \eta , \alpha < 1$.
† Assuming $F_a^i$ = sigmoid function.
```

3. Statement of the Problem

The main issue in robot control is to solve for the dynamic nonlinear equations of motion. If the manipulator motion is speeded up or slowed down due to the change of the arm payload then the robot arm is unable to follow the planned trajectory. Therefore, the generalized torques/forces must be computed for all joints of the arm at each control cycle, which involves the solution of the inverse dynamics procedure. The dynamic model of the robot is a set of second order differential equations defined as follows

$$\boldsymbol{\tau}(t) = \mathbf{D}(\boldsymbol{\theta}(t))\,\ddot{\boldsymbol{\theta}}(t) + \mathbf{H}(\boldsymbol{\theta}(t),\dot{\boldsymbol{\theta}}(t)) + \mathbf{C}(\boldsymbol{\theta}(t)) \tag{1}$$

or alternatively,

$$\tau_i = \sum_{j=1}^{N} D_{ij}\ddot{\theta}_j + \sum_{j=1}^{N}\sum_{k=1}^{N} H_{ijk}\,\dot{\theta}_j\,\dot{\theta}_k + C_i \tag{2}$$

$$, \quad i = 1,2,\ldots,N$$

where,

$\boldsymbol{\tau} \equiv$ Torque values,

$\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$, $\ddot{\boldsymbol{\theta}} \equiv$ Position, velocity and acceleration of each joint,

$D_{ij} \equiv$ Effective and coupling inertias,

$H_{ijk} \equiv$ Centripetal and coriolis forces,

$C_i \equiv$ Gravity loading, and

$N \equiv$ Number of joints of a given manipulator.

The computations of (eqn.1) are considered to be very difficult to carry out in real-time applications, especially once augmented with other control requirements [Zalzala and Morris 1989b]. It has been estimated to occupy 36% of the total controller execution time [Orin 1984]. Several simplifications have been suggested to cut down the computational burden [Bejczy 1974, Paul,Rong and Zhang 1983]. However, due to the presence of highly effective coupling in these nonlinear equations, experimental results have shown that such simplifications affect unacceptably the accuracy of tracking [Tang and Tourassis 1989].

Therefore, one fairly recent approach has been the use of modern VLSI technology to accommodate for the computational complexities in terms of distributed parallel structures [Zalzala and Morris 1989g, Lathrop 1985, Lee and Chang 1986].

4. Symbolic Representation of The Dynamic Equations

The Lagrange-Euler approach to the formulation of the dynamic equations stated above is quite structured. The torque value of each joint is expressed as the sum of three main terms, namely : the *acceleration-related term* (τ_A), the *velocity-related term* (τ_V) and the *gravity term* (τ_G). Hence, (eqn.1) can be rewritten as

$$\boldsymbol{\tau}^i_{total} = \boldsymbol{\tau}^i_A + \boldsymbol{\tau}^i_V + \boldsymbol{\tau}^i_G \tag{3}$$

where,

$$\tau^i_A = \sum_{j=1}^{N} D_{ij}\ddot{\theta}_j \tag{4}$$

$$\tau_V^i = \sum_{j=1}^{N} \sum_{k=1}^{N} H_{ijk}\, \dot{\theta}_j\, \dot{\theta}_k \tag{5}$$

and,

$$\tau_G^i = C_i \tag{6}$$

One method of representing these equations is in a symbolic form. In such an approach, all unnecessary computations at lower levels of the formulation are omitted (e.g. multiplications by 1s and 0s, and additions to 0s), thus producing a more compact code. In addition, all programming structures would be eliminated, reducing a large amount of unnecessary execution time. It should be noted, however, that due to different configurations of serial-links arms, a symbolic representation should be produced for each robot arm individually. Several symbolic representations have been described in the literature for the Lagrange-Euler formulation of the PUMA manipulator [Paul,Rong and Zhang 1983, Tarn,Bejczy,Yun and Ding 1986].

A comparison between several inverse dynamics formulations for the PUMA arm is shown in table (2), which clearly indicates the superiority of symbolic computations. The relevant programs has been executed on the *Sun* 3/50 system with a floating point accelerator [Sun 1986].

Table (2) : Computational Complexity of the Inverse Dynamics

Inverse Dynamics Procedure	Execution time (seconds)
Classical Lagrangian (no simplifications)	0.9
Generalized D'Alembert	0.096
Recursive Lagrangian	0.054
Symbolic Lagrangian	0.031

Folding out individual terms of (eqn.2) for a six d.o.f. manipulator, the following representation could be found for each of the terms defined by (eqns.4-6) :

A. The acceleration-related term :

$$\mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} \\ & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} \\ & & D_{33} & D_{34} & D_{35} & D_{36} \\ & Symm- & & D_{44} & D_{45} & D_{46} \\ & etric & & & D_{55} & D_{56} \\ & & & & & D_{66} \end{bmatrix} \tag{7}$$

B. The velocity-related term :

$$\mathbf{H}_1 = \begin{bmatrix} 0 & H_{112} & H_{113} & H_{114} & H_{115} & H_{116} \\ & H_{122} & H_{123} & H_{124} & H_{125} & H_{126} \\ & & H_{133} & H_{134} & H_{135} & H_{136} \\ & Symm- & & H_{144} & H_{145} & H_{146} \\ & etric & & & H_{155} & H_{156} \\ & & & & & H_{166} \end{bmatrix} \tag{8}$$

$$\mathbf{H}_2 = \begin{bmatrix} \underline{-H_{112}} & 0 & H_{213} & H_{214} & H_{215} & H_{216} \\ & 0 & H_{223} & H_{224} & H_{225} & H_{226} \\ & & H_{233} & H_{234} & H_{235} & H_{236} \\ & Symm- & & H_{244} & H_{245} & H_{246} \\ & etric & & & H_{255} & H_{256} \\ & & & & & H_{266} \end{bmatrix} \tag{9}$$

$$\mathbf{H}_3 = \begin{bmatrix} \underline{-H_{113}} & \underline{-H_{213}} & 0 & H_{314} & H_{315} & H_{316} \\ & \underline{-H_{223}} & 0 & H_{324} & H_{325} & H_{326} \\ & & 0 & H_{334} & H_{335} & H_{336} \\ & Symm- & & H_{344} & H_{345} & H_{346} \\ & etric & & & H_{355} & H_{356} \\ & & & & & H_{366} \end{bmatrix} \tag{10}$$

$$\mathbf{H}_4 = \begin{bmatrix} \underline{-H_{114}} & \underline{-H_{214}} & \underline{-H_{314}} & 0 & H_{415} & H_{416} \\ & \underline{-H_{224}} & \underline{-H_{324}} & 0 & H_{425} & H_{426} \\ & & \underline{-H_{334}} & 0 & H_{435} & H_{436} \\ & \mathit{Symm-} & & 0 & H_{445} & H_{446} \\ & \mathit{etric} & & & H_{455} & H_{456} \\ & & & & & H_{466} \end{bmatrix} \tag{11}$$

$$\mathbf{H}_5 = \begin{bmatrix} \underline{-H_{115}} & \underline{-H_{215}} & \underline{-H_{315}} & \underline{-H_{415}} & 0 & H_{516} \\ & \underline{-H_{225}} & \underline{-H_{325}} & \underline{-H_{425}} & 0 & H_{526} \\ & & \underline{-H_{335}} & \underline{-H_{435}} & 0 & H_{536} \\ & \mathit{Symm-} & & \underline{-H_{445}} & 0 & H_{546} \\ & \mathit{etric} & & & 0 & H_{556} \\ & & & & & H_{566} \end{bmatrix} \tag{12}$$

$$\mathbf{H}_6 = \begin{bmatrix} \underline{-H_{116}} & \underline{-H_{216}} & \underline{-H_{316}} & \underline{-H_{416}} & \underline{-H_{516}} & 0 \\ & \underline{-H_{226}} & \underline{-H_{326}} & \underline{-H_{426}} & \underline{-H_{526}} & 0 \\ & & \underline{-H_{336}} & \underline{-H_{436}} & \underline{-H_{536}} & 0 \\ & \mathit{Symm-} & & \underline{-H_{446}} & \underline{-H_{546}} & 0 \\ & \mathit{etric} & & & \underline{-H_{556}} & 0 \\ & & & & & 0 \end{bmatrix} \tag{13}$$

C. The gravity term :

$$\mathbf{C} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix} \tag{14}$$

The symbolic representation for each term shown in (eqns.7-14) are adapted from [Tarn,Bejczy,Yun and Ding 1986] for the PUMA 560 manipulator. A full model of the arm is included along with a particular gripper and a certain load [Tarn,Bejczy,Han and Yun 1985]. No simplifications were assumed in the model of the arm used.

5. The Distributed Formulation

The form of (eqns.7-14) readily lends itself to a distributed recasting, introducing certain concepts in parallel processing. In addition, all terms shown underlined are redundant and can be obtained from previously computed elements of the equations. However, using a general purpose multi-processing system would introduce tremendouse difficulties for a MIMD structure. In practical implementations, the communications overhead, which is certainly extensive in this case, could severely disrupt the efficiency of the distributed algorithm. Although theoretical formulations usually do not take proper account of this problem [Luh and Lin 1982], it has been proven to have its effect on actual multiprocessor systems [Zalzala and Morris 1989a].

Nevertheless, the use of sophisticated and specialised VLSI structures promises to present a solution for such a problem. If each of these terms were to be placed on separate computational nodes which each have the ability of intercommunication, and further a suitable efficient distributed algorithm was to be provided, satisfactory results would be expected.

The specialized computational nodes required for each term of the dynamic equations are shown in table (3).

Table (3) : Required Computational Nodes

Dynamic equations term	Number of nodes
Acceleration-Related	20
Velocity-related	67
Gravity	5

6. A Neural Controller Prospect

The distributed structure described in the previous section is fully accommodated for in a neural network approach. In this section, adaptation of the back-propagation algorithm for this purpose is shown, and further a multi-layered neural structure for the full dynamic model of (eqn.3) is presented. The general organization of a real-time neural controller is shown in figure (3), where the adaptation of the neural network depends upon the sensory feedback information in each control cycle. Hence, if the model of the actual robot varies from that predicted by the inverse dynamics algorithm solved by the controller, the network weights are adjusted so as to cancel such a difference. Thus, as time progresses, the controller should correct any errors occuring during tracking the desired motion.

As shown in figure (3), assuming the robot arm is following some desired trajectory, the parameters of motion (i.e. position θ_i, velocity $\dot{\theta}_i$ and acceleration $\ddot{\theta}_i$), for all joints and within each control cycle, are fed to the controller producing a desired torque input for each of the motors. The actual applied values of torques are detected, and the errors are propagated back, adjusting the neural controller. Once a second set of motion parameters is applied, the output of the controller is expected to be more likely to accurately direct the motion towards that desired. The repetitive execution of such a process would cause the controller to learn a new model under which the required motion should be performed.

Nevertheless, to provide for the errors in the torque values as an input to the second phase of the back-propagation rule, our approach for the simulation was to assign another similar neural network (a torque computations module) to run simultaneously with the controller. However, a torque measurement device could be used in a practical system, providing the actual values of all joint motor torques.

6.1. Applying the Back-propagation Learning Rule

To implement the proposed distributed algorithm, the back-propagation rule must run on several hidden layers, with nodes that are sparsely connected, according to the needs of the mathematical formulation. Refering to table (1), the following is assumed:

- Initial weights for all nodes of the network are set to unity, hence

$$W_{ij} = 1 \quad , i=1,2,...,Nd(L) \quad , j=1,2,...,Nd(L-1) \tag{15}$$

where $Nd(X)$ denotes the number of connected nodes in layer X.

- All bias values are set to zero, thus

$$\beta_k = 0 \quad ,k=1,2,...Nn \tag{16}$$

where Nn is the total number of nodes in the network.

- The activation function is chosen to be a sigmoid of the form [Scott Stornetta and Huberman 1987],

$$S_i = F_a^i\,(F_c^i\,,\,\beta_i) = -\frac{1}{2} + \frac{1}{1 + e^{(\beta_i - F_c^i)}} \tag{17}$$

Although several other forms of activation functions were suggested

[Lippmann 1987, Gibson,Siu and Cowan 1989], the above choice is found to be adequate for this application. The characteristics of such a function is illustrated by the graph of figure (4).

□ The combining function is intended to form a relation between all inputs to a specific node and their weights. No restrictions on the form of such a function have been made, although a purely linear function of the form

$$F_c^i = \sum_j I_j W_{ij} \tag{18}$$

has been suggested [McClelland and Rumelhart 1988].

For this application, the combining function is taken to have a nonlinear nature, and is set as one term of (eqns.7-14) for each neuron in the network. Hence, for each node,

$$F_c^i = fn\ (P) \tag{19}$$

and,

$$P = S_j^{-1}\ (J_1\ , J_2\ ,\ \cdots\ , J_j) \tag{20}$$

where,

$$J_j = I_j\ W_{ij} \tag{21}$$

and S_i^{-1} is the inverse-sigmoid function defined as

$$S_i^{-1} = \beta_i - Ln \left[\frac{1}{F_a^i + \frac{1}{2}} - 1 \right] \tag{22}$$

Hence, the main task of each node is to execute one part of the nonlinear equations of motion, and transmit its output to the nodes of the next hidden layer requiring it.

6.2. Network Construction

Since the total torques evaluation process could be subdivided into three main parts as indicated by (eqn.3) earlier, the neural network is to be constructed accordingly. Initially, an input layer of 18 nodes was constructed, where six nodes were used for each set of the parameters of motion (i.e. θ_i, $\dot{\theta}_i$ and $\ddot{\theta}_i$). In addition, an output layer of 6 nodes was considered, yielding the required motor inputs for a 6 d.o.f. robot. Three sets of hidden layers were designated for the distributed algorithm, each set accommodating for one term of the dynamic equations :

A. Acceleration-related term : two hidden layers are required with 20 and 6 nodes in each, respectively.

B. Velocity-related term : three hidden layers are required with 67, 6 and 6 nodes in each, respectively.

C. Gravity term : only one hidden layer is required with 5 nodes.

The function of each set of hidden layers is to produce one portion of the torque values, which are then collected and added together by nodes of the output layer. The structure of the proposed network is shown in figure (5).

The function of each hidden layer in the above sets is as follows :

1. Computing τ_A :
   - 1.1. Hidden layer #1 : each node of this layer computes one element of the inertial matrix (eqn.7).
   - 1.2. Hidden layer #2 : each node multiplies one row of the inertial matrix (computed by the previous layer) by the acceleration vector of all joints, thus producing τ_A for each joint of the PUMA given by (eqn.4).
2. Computing τ_V :
   - 2.1. Hidden layer #1 : each node of this layer computes one element of the coriolis and centripetal force expressed by (eqns.8-13).
   - 2.2. Hidden layer #2 : each node multiplies one matrix of the coriolis and centripetal forces related to one joint (computed by the previous layer) by the velocity vector of all joints.
   - 2.3. Hidden layer #3 : each node multiplies the output vector computed by the previous layer times the velocity vector of all joints, thus producing τ_V for each joint of the PUMA given by (eqn.5).
3. Computing τ_G :
   - 3.1. Hidden layer #1 : each node of this layer computes one element of the position-related gravity term expressed by (eqn.14).

As was mentioned earlier, the connections amongst nodes throughout the network is sparse, depending upon the needs of the mathematical formulation. The network of the first set of hidden layers describing the computation of the acceleration-related torque values is shown in figure (6). Programming has been performed in the C language, where the source code of each of the nodes of the first hidden layer of the acceleration-related term is shown in *Appendix (A)*.

7. Mathematical Justification of the Proposed Model

The main aim of the proposed neural model has been two fold :

a. Cutting down the computational complexities by introducing a distributed algorithm for the involved equations.

b. Providing a learning scheme through which adaptation to any changes in the environment could be accommodated for.

The purpose of this section is to provide an explanation of how the learning process progresses and the methodology by which the robot model is adapted according to need. Each node of the first hidden layer for each of the terms of (eqn.3) contains one element of the nonlinear dynamic equations of the arm, which we denote in general as (Γ^1). These elements are functions of the robot joint positions and the arm model parameters. Hence, for all three terms,

$$\Gamma^1 = fn\ (\ \theta\ ,\ \chi\ ) \tag{23}$$

where χ denotes the model parameters. In addition, nodes of other hidden layers in the network would be a function of Γ^1 and another parameter of motion. Hence, for the acceleration-related term,

$$\Gamma_A^2 = fn\ (\ \Gamma_A^1\ ,\ \ddot{\theta}\ ) = fn\ (\ \theta\ ,\ \chi\ ,\ \ddot{\theta}\ ) \tag{24}$$

and similarily for the velocity related term,

$$\Gamma_V^2 = fn\ (\ \Gamma_V^1\ ,\ \dot{\theta}\ ) = fn\ (\ \theta\ ,\ \chi\ ,\ \dot{\theta}\ ) \tag{25}$$

Adaptation is achieved by changing the weights of particular nodes containing functions of the form given in (eqn.23). In (eqn.23), the position parameter ($\boldsymbol{\theta}$) must be kept constant as it represents the required target. Hence, adaptation can only involve changing the set of parameters (χ). Such a procedure is similar to a *parameters estimation method,* where the weights are changed in each control cycle, thereby adapting the model parameters according to the errors produced by inaccurate tracking.

Consequently, the change in weights of the inputs of each node accommodating a function of the form of (eqns.24,25) can be interpreted as adapting the *gain* values applied to the $\ddot{\theta}$ and $\dot{\theta}$ parameters, respectively. Hence, a dynamic determination of such gains is being made in each control interval. The weights of the other parameters involved in (eqns.24,25) (i.e. Γ_A^1 and Γ_V^1) would again have influence upon the outputs of the first hidden layer.

Therefore, the proposed neural controller is seen as having the ability to perform parameter estimation of the robot arm model during each control cycle, and further to perform a dynamic adjustment of the gain values for the parameters of motion.

8. Simulation Results

Simulation results will now be presented for a pre-defined motion of a PUMA 560 robot arm. Referring to figure (3), the robot arm can be simulated using the Direct Dynamics Algorithm [Walker and Orin 1982, Swartz 1984], which produces the actual parameters of the motion executed by the arm. The output of this procedure is fed into a controller similar to that used in the learning process. However, this second controller has no learning abilities, but is dedicated to producing the actual torque values applied by the motors. As a case study, six joint trajectories (one for each joint of the PUMA) have been considered [Zalzala and Morris 1988, Zalzala and Morris 1989f], for which a total of 3144 control cycles are to be executed. Experiments were conducted so as to choose the best values for the learning factor (η), and the momentum factor (α), where a separate independent value of each was considered for each hidden layer of the network. This was found to give much better results in terms of minimising the tracking errors than to having one common factor for all layers. The values of these factors are shown in table (4) for the acceleration-related term.

Table (4)
Learning and Momentum Factors
for the Acceleration-related Term

Factor	Layer		
	First	Second	
		Output of 1st Layer	$\ddot{\theta}$ values
Learning (η)	0.02	0.03	0.25
Momentum (α)	0.3	0.4	0.4

Applying the learning rule described in the previous sections, errors in tracking the desired motion were significantly reduced. The time history of the reduction of error is shown in figures (7) and (8) for joints 1 and 6 of the PUMA arm. The adjusted weights of each layer of the acceleration-related term are shown in tables (5) and (6) for illustration.

Table (5)
Adjusted Weights of the Acceleration-related Term
(First Hidden Layer)

Node #	Weight #				
	W_1	W_2	W_3	W_4	W_5
1	0.999980	1.001077	0.999078	0.999957	1.000362
2	1.001512	1.000761	0.998835	0.999031	1.000026
3	1.000005	0.999989	1.000000	0.999996	0.999999
4	1.000000	0.999998	1.000000	1.000000	1.000000
5	1.000000	1.000000	1.000000	1.000000	1.000000
6	1.000000	1.000000	1.000000	1.000000	1.000000
7	1.001596	1.000031	1.000229	1.000719	NA †
8	1.000678	0.999902	1.000126	1.000286	NA
9	1.000002	0.999999	1.000000	1.000000	NA
10	1.000000	1.000000	1.000000	1.000000	NA
11	1.000000	1.000000	1.000000	1.000000	NA
12	1.000010	1.000005	1.000000	NA	NA
13	1.000000	1.000000	1.000000	NA	NA
14	1.000000	1.000000	1.000000	NA	NA
15	1.000000	1.000000	1.000000	NA	NA
16	1.000000	1.000000	NA	NA	NA
17	1.000000	1.000000	NA	NA	NA
18	1.000000	NA	NA	NA	NA
19	1.000000	NA	NA	NA	NA
20	NA	NA	NA	NA	NA

† NA = Not Applicable

Table (6)
Adjusted Weights of the Acceleration-related Term
(Second Hidden Layer)

Weight #	Node #					
	1	2	3	4	5	6
W_1	1.003323	0.999255	1.000000	1.000000	1.000000	1.000000
W_2	1.000049	1.005180	1.000003	1.000000	1.000000	1.000000
W_3	0.999982	1.001536	1.000008	1.000000	1.000000	1.000000
W_4	0.999998	1.000004	1.000000	1.000000	1.000000	1.000000
W_5	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
W_6	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
W_7	0.997999	1.012662	0.999395	1.000000	1.000000	1.000000
W_8	1.001167	0.993676	1.000385	1.000000	1.000000	1.000000
W_9	0.998926	1.030629	1.000777	1.000000	1.000000	1.000000
W_{10}	1.002072	0.949169	0.998434	0.999999	1.000000	1.000000
W_{11}	1.000061	0.968776	0.999153	1.000000	1.000000	1.000000
W_{12}	0.999712	1.001576	0.999774	1.000000	1.000000	1.000000

One side result of great importance was detected during the simulations. It is a well known fact that the coriolis, cetripetal and gravitational forces play a large role in preserving the accuracy of motion, and ignoring such terms in practical implementations was found to cause considerable tracking errors [An,Atkeson,Griffiths and Hollerbach 1989]. However, ignoring both the velocity-related and the gravity terms in the neural controller lead to the acceleration-related network adaptation to the new situation, and the errors were reduced nevertheless.

The simulation was performed on a *Sun* 3/50 workstation, employing the *C* programming language.

9. Practical VLSI Implementation : A Performance Analysis

In this section, an analysis of applying the proposed neural controller as an actual multiprocessor system is conducted. Two options will be considered. First, a general purpose system, where the INMOS T800 transputer is taken as a practical example. Second, specialised VLSI structures are assessed for such an implementation.

9.1. The T800 Transputer Based System

The INMOS transputer (T800) is a fast, one-board computer that can be used as the main block in building a multiprocesser system [INMOS 1988a, INMOS 1988b]. The T800 machine has been successfully used in many scientific and engineering applications, including real-time robot control [Zalzala and Morris 1989a, Zalzala and Morris 1989g]. Nevertheless, the use of such a machine to implement the robot neural controller has its restrictions, as follows:

- the concept of assigning a full transputer board to each of the neural network nodes is utterly impractical. Such an implementation would require a total of 134 transputers to accommodate for it, and many of the transputers would be idle for most of the time considering the difference in the computational complexities of all nodes. Hence, the overall utilisation of the VLSI structure would be severely low. In addition, the cost-effectiveness of such a system becomes questionable.
- The nodes of the proposed neural network require for certain cases a maximum of 13 channels to accommodate for the communications required by other neighbouring nodes. This fact highlights the hardware limitation imposed by the transputer design, which allows the device only four channels for sending and receiving data. This in turn would lead to a massive overhead in communications.

However, certain solutions can be proposed for the above two problems, as shown next.

9.1.1. The Job Scheduling Procedure

Since the use of a single transputer for each node of the network had been seen as impractical, the other alternative would be to distribute the groups of jobs of all nodes on several transputers in an optimum scheduling scheme, thus providing for a high utilisation of the devices, in addition to minimising the execution time.

To accomplish such a task, the computational complexities of each of the neural nodes have been assessed according to the capabilities and speed of the T800 transputer. According to the T800 data sheets, the computational time of both an addition operation (T_{add}) and a multiplication operation (T_{mul}) is 350 nsec and 550 nsec, respectively. Hence, the computational requirements of each of the neural nodes could be computed as †

$$CW = N_{add}T_{add} + N_{mul}T_{mul} \tag{26}$$

where N_{add} and N_{mul} denote the number of addition and multiplication operations within a node. The calculated time is to be defined as the *computational weight* (CW), of that node. Furthermore, a *normalised computational weight* (nCW) is defined for each node as

$$nCW = \frac{CW}{\underset{j=1}{\overset{Nn}{MAX}}\, CW_j} \tag{27}$$

The resultant calculations are shown in tables (7-9) for all three terms composing the torque values.

The scheduling procedure will now be illustrated for the acceleration related term, considering the results of table (7), while similar arguments could be made for other terms. From table (7), the element D_{11} has the largest computational burden (i.e. nCW=1). Hence, it is desirable to compute all other elements in a similar time. Thus, the execution of node 1 is to be performed on an independent transputer. Since all nodes in the first hidden layer are independent from each other, they are to be collected in groups using the bin-packing method [Hu 1982] so as

$$\sum_{i=1}^{m} nCW_i \leq 1 \tag{28}$$

for each transputer machine, where m denotes the number of jobs to be placed on that particular transputer. However, the utilisation of each of the processors should be kept as high as possible. One possible distribution is shown in figure (9), where a total of 3 transputers are needed. In addition, the nodes of layer #2 are included as shown. The utilisation of each processor is shown in table (10). The evaluation of the trigonometric functions involved in each term is domonated by those of the node with the largest weight factor in the layer (e.g. node #1 containing element D_{11} in the acceleration-related term). All similar computations could be performed on each transputer seperately for all the nodes scheduled on it. An evaluation of the trigonometric functions required by the element computing τ_A are shown in table (7).

† Trigonometric functions are not included in this evaluation

Table (7)
Computational Complexity of the Acceleration-related Term

Term	(*) Operations	(+) Operations	Computing Weight (*CW*) (μsec)	Normalised Weight (*nCW*)	Trigonometric Evaluations			
					Sin	Cos	(+)	(*)
D_{11}	437	183	304.4	1.00	6	6	8	26
D_{12}	159	69	111.6	0.37	5	5	6	16
D_{13}	97	54	72.3	0.24	3	3	6	16
D_{14}	87	39	61.5	0.20	3	4	4	10
D_{15}	52	20	35.6	0.12	3	4	4	7
D_{16}	32	12	21.8	0.07	4	5	4	4
D_{22}	86	58	67.6	0.22	4	4	0	9
D_{23}	84	54	65.1	0.21	4	4	1	10
D_{24}	37	20	27.4	0.09	4	3	0	3
D_{25}	32	18	23.9	0.08	4	4	0	3
D_{26}	20	8	13.8	0.05	4	4	0	0
D_{33}	47	36	38.5	0.13	3	3	1	9
D_{34}	29	14	20.9	0.07	3	3	0	3
D_{35}	23	12	16.9	0.06	3	3	0	3
D_{36}	13	5	8.9	0.03	3	3	0	0
D_{44}	8	6	6.5	0.02	2	1	0	3
D_{45}	4	1	2.6	0.01	2	0	0	2
D_{46}	2	0	1.1	0.00	0	1	0	0
D_{55}	4	2	2.9	0.01	1	1	0	2
D_{56}	0	0	0.0	0.00	0	0	0	0
D_{66}	1	0	0.6	0.00	0	0	0	0

Table (8)
Computational Complexity of the Velocity-related Term

Term	(*) Operations	(+) Operations	Computing Weight (CW) (μsec)	Normalised Weight (nCW)
H_{112}	244	114	170	1.00
H_{113}	192	105	140	0.82
H_{114}	197	77	135	0.78
H_{115}	237	92	163	0.93
H_{116}	153	52	102	0.59
H_{122}	120	60	87	0.50
H_{123}	101	50	73	0.42
H_{124}	47	28	36	0.20
H_{125}	54	23	38	0.22
H_{126}	35	16	25	0.14
H_{133}	105	50	75	0.43
H_{134}	49	28	37	0.21
H_{135}	55	23	38	0.22
H_{136}	33	15	23	0.13
H_{144}	66	27	46	0.26
H_{145}	49	23	35	0.20
H_{146}	30	14	21	0.12
H_{155}	42	18	29	0.17
H_{156}	31	11	21	0.12
H_{166}	29	9	19	0.11

Table (8) : Continued

Term	(*) Operations	(+) Operations	Computing Weight (*CW*) (μsec)	Normalised Weight (*nCW*)
H_{213}	0	0	0	0.00
H_{214}	85	41	61	0.35
H_{215}	45	21	32	0.18
H_{216}	42	17	29	0.17
H_{223}	36	20	27	0.15
H_{224}	44	23	32	0.19
H_{225}	36	19	26	0.15
H_{226}	22	11	16	0.09
H_{233}	34	20	26	0.15
H_{234}	45	24	33.2	0.19
H_{235}	38	19	27.6	0.16
H_{236}	26	11	18.2	0.10
H_{244}	33	18	24.5	0.14
H_{245}	29	14	20.9	0.12
H_{246}	17	8	12.2	0.07
H_{255}	30	14	21.4	0.12
H_{256}	19	8	13.3	0.08
H_{266}	18	7	12.4	0.07
H_{314}	80	38	57.3	0.33
H_{315}	53	20	36.2	0.21
H_{316}	34	15	24.0	0.14

Table (8) : Continued

Term	(*) Operations	(+) Operations	Computing Weight (CW) (μsec)	Normalised Weight (nCW)
H_{324}	36	18	26.1	0.15
H_{325}	30	13	21.1	0.12
H_{326}	17	9	12.5	0.07
H_{334}	35	19	25.9	0.15
H_{335}	38	13	25.5	0.15
H_{336}	20	8	13.8	0.08
H_{344}	37	14	25.3	0.15
H_{345}	25	11	17.6	0.10
H_{346}	18	7	12.4	0.07
H_{355}	21	8	14.4	0.08
H_{356}	16	7	11.3	0.06
H_{366}	11	4	7.5	0.04
H_{415}	28	15	20.7	0.12
H_{416}	12	4	8.0	0.05
H_{425}	17	8	12.2	0.07
H_{426}	10	5	7.3	0.04
H_{435}	18	12	14.1	0.08
H_{436}	8	3	5.5	0.03
H_{445}	7	5	5.6	0.03
H_{446}	4	1	2.6	0.01
H_{455}	4	1	2.6	0.01

Table (8) : Continued

Term	(*) Operations	(+) Operations	Computing Weight (*CW*) (µsec)	Normalised Weight (*nCW*)
H_{456}	5	4	4.2	0.02
H_{466}	0	0	0.0	0.00
H_{516}	14	5	9.5	0.05
H_{526}	8	3	5.5	0.03
H_{536}	8	3	5.5	0.03
H_{546}	5	4	4.2	0.02
H_{556}	3	1	2.0	0.01
H_{566}	0	0	0.0	0.00

Table (9)
Computational Complexity of the Gravity Term

Term	(*) Operations	(+) Operations	Computing Weight (*CW*) (µsec)	Normalised Weight (*nCW*)
C_1	0	0	0.0	0.00
C_2	42	27	32.6	1.00
C_3	34	19	25.4	0.78
C_4	24	7	15.7	0.48
C_5	16	8	11.6	0.36
C_6	12	4	8.0	0.25

Table (10)
Processors Utilisation

Transputer	Utilisation
T1	1.00
T2	1.00
T3	0.98

9.1.2. The Problem of Communications

The scheduling procedure illustrated in the previous section divides the neurons in each hidden layer into groups, each placed on one transputer. The communications between all three transputers needed to implement the acceleration-related term is shown in figure (10). A *host* transputer would act as a supervisor representing the *input* and *output* layers of the network. One link is assigned for the communications between transputers $P1$ and $P2$. However, it is clear from figure (9) that more communications occur between transputers $P2$ and $P3$. Therefore, two communication links are assigned for the job.

The number of nodes in the neural network presented is relatively small in comparison to other applications. Therefore, if a much larger number of nodes exists, more sophisticated methods would be required to manage the intercommunication tasks [Kamanger,Duderstadt and Smith 1989, Beynon and Dodd 1987].

9.2. Specialised VLSI Structures

The alternative to a general purpose multiprocessor system is the design of specialised VLSI structures which may be adequate to perform the jobs required by each node. In addition, the availability of direct link communications between each two neurons would greatly enhance the performance of the algorithm. Several architectures are currently available [Karna and Breen 1989, Kamanger,Duderstadt and Smith 1989], but these are inadequate in many respects and current research is directed towards producing more feasable designs [Habib and Akel 1989]. This subject is beyond the scope of this paper, and has been included here for completion.

10. Acknowledgement

The symbolic representation of the dynamic equations was coded in the C programming language by A. K. Al-Dulaimi.

11. Conclusion

A neural-based controller has been presented for the control of robot manipulators. The contribution offered by this paper is two fold. First, a distributed algorithm has been presented which employs a symbolic representation of the Lagrangian dynamic equations, thus providing for a fast feedforward controller. Second, the concepts of learning using artificial neural networks have been used to give the controller an ability to adapt to changes in the environment and to uncertainties in the implemented model of the arm. In addition, the feasability of implementing the proposed algorithm on a general purpose multiprocessor system, namely a T800 transputer network, has been illustrated. The primary benefit of augmenting both the conventional theory of robot control with the concepts of artificial neural networks is to produce a fast self-organizing robot controller. Such a design promises to become a strong opponent to the conventional adaptive robot controllers, especially when real-time applications are involved.

References

[Sun 1986].
SUN, *Floating-Point Programmer's Guide for the Sun Workstation,* (Sun Microsystems Inc., 1986).

[INMOS 1988a].
INMOS LTD., "IMS T800 Architecture", Technical Note #6, (1988) .

[INMOS 1988b].
INMOS LTD., *Transputer Development System,* (Prentice-Hall, 1988).

[Aleksander 1989].
ALEKSANDER, I., (ED.), *Neural Computing Architectures : The Design of Brain-Like Machines,* (North Oxford Academic, 1989).

[An,Atkeson,Griffiths and Hollerbach 1989].
AN, C. H., ATKESON, C. G., GRIFFITHS, J. D., AND HOLLERBACH, J. M., "Experimental Evaluation of Feedforward and Computed Torque Control ", *IEEE Trans. Robotics and Automation,* Vol. RA-5, No. 3, pp. 368-73, (1989).

[Angus 1989].
ANGUS, J. E., "On the Connection Between Neural Network Learning and Multivariate Nonlinear Least Squares Estimation ", *Int. J. Neural Networks: Research and Applications,* Vol. 1, No. 1, pp. 42-46, (1989).

[Arbib 1987].
ARBIB, M. A., *Brains, Machines and Mathematics,* (Springer-Verlag, 1987).

[Bassi and Bekey 1989].
BASSI, D. F. AND BEKEY, G. A., "Decomposition of Neural Network Models of Robot Dynamics: A Feasability Study", In *Simulation and AI, Proc. Western Multiconference,* (San Diego,1989) , pp. 8-13.

[Bejczy 1974].
BEJCZY, A. K., "Robot Arm Dynamics and Control ", *JPL Labs. Tech. Memo #33-669,* (1974).

[Beynon and Dodd 1987].
BEYNON, T. AND DODD, N., "The Implementation of Multi-Layer Perceptrons on Transputer Networks", Report RIPRREP/1000/13/87, (Research Initiative in Pattern Recognition, Royal Signals and Radar Establishment,Malvern, WR14 3PS, United Kingdom,1987) .

[Craig 1986].
CRAIG, J. J., *Introduction to Robotics : Mechanics and Control,* (Addison-Wesley, 1986).

[Daunicht 1989].
DAUNICHT, W. J., "Control of Manipulators by Neural Networks ", *IEE Proceedings, Pt. E,* Vol. 136, No. 5, pp. 395-9, (1989).

[Dubowsky and DesForges 1979].
DUBOWSKY, S. AND DESFORGES, D. T., "The Application of Model Reference Adaptive Control to Robotic Manipulators ", *Trans. ASME, J. Dyn. Syst., Meas. and Cntrl.,* Vol. 101, pp. 193-200, (1979).

[Elsley 1987].
ELSLEY, R., "A Learning Architecture for Control Based on Back-propagation Neural Networks", In *Proc. IEEE Int. Conf. Neural Networks,* Vol. 2, (1987) , pp. 587-94.

[Fu,Gonzalez and Lee 1987].
FU, K. S., GONZALEZ, R. C., AND LEE, C. S. G., *Robotics : Control, Sensing, Vision and*

Intelligence, (McGraw Hill,New York, New York, 1987).

[Gibson,Siu and Cowan 1989].
GIBSON, G. J., SIU, S., AND COWAN, C. F. N., "Application of Multilayer Perceptrons as Adaptive Channel Equalisers", In *Proc. IFAC Symp. Adaptive Systems in Control and Signal Processing*, Vol. 1, (1989) , pp. 323-328.

[Guez,Eilbert and Kam 1987].
GUEZ, A., EILBERT, J., AND KAM, M., "Neuromorphic Architecture for Adaptive Robot Control: A Preliminary Analysis", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 4, (1987) , pp. 567-72.

[Guez and Ahmad 1988].
GUEZ, A. AND AHMAD, Z., "Solution to the Inverse Kinematics Problem in Robotics by Neural Networks", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 2, (1988) , pp. 617-23.

[Guez,Eilbert and Kam 1988].
GUEZ, A., EILBERT, J. L., AND KAM, M., "Neural Network Architecture for Control ", *IEEE Control Systems Magazine*, pp. 22-5, (1988).

[Habib and Akel 1989].
HABIB, M. K. AND AKEL, H., "A Digital Neuron-Type Processor and its VLSI Design ", *IEEE Trans. Circuits and Systems*, Vol. 36, No. 5, pp. 739-46, (1989).

[Hu 1982].
HU, T. C., *Combinatorial Algorithms*, (Addison-Wesley, 1982).

[Scott Stornetta and Huberman 1987].
SCOTT STORNETTA, W., HUBERMAN, B. A., "An Improved Three-Layer Back Propagation Algorithm", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 2, (1987) , pp. 637-43.

[Jorgensen 1987].
JORGENSEN, C. C., "Neural Network Representation of Sensor Graphs in Autonomous Robot Path Planning", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 4, (1987) , pp. 507-15.

[Josin,Charney and White 1988].
JOSIN, G., CHARNEY, D., AND WHITE, D., "Robot Control Using Neural Networks", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 2, (1988) , pp. 625-31.

[Kamanger,Duderstadt and Smith 1989].
KAMANGER, F. A., DUDERSTADT, R. A., AND SMITH, J. O., "Implementing the Back-Propagation Algorithm on the Meiko Parallel Computing Surface", In *Proc. Int. Conf. Applications of Transputers*, (23-25 August, University of Liverpool, United Kingdom (to appear),1989) .

[Karna and Breen 1989].
KARNA, K. N. AND BREEN, D. M., "An Artificial Neural Networks Tutorial: Part 1 - Basics ", *Int. J. Neural Networks: Research and Applications*, Vol. 1, No. 1, pp. 4-23, (1989).

[Kawato,Uno and Isobe 1987].
KAWATO, M., UNO, Y., AND ISOBE, M., "A Hierarchial Model for Voluntary Movement and its Application to Robotics", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 4, (1987) , pp. 573-82.

[Kawato,Uno,Isobe and Suzuki 1988].
KAWATO, M., UNO, Y., ISOBE, M., AND SUZUKI, R., "Hierarchical Neural Network Model for Voluntary Movement with Application to Robotics ", *IEEE Control Systems Magazine*, pp. 8-16, (1988).

[Koivo and Guo 1983].
KOIVO, A. J. AND GUO, T. H., "Adaptive Linear Controller for Robotic Manipulators ", *IEEE Trans. Automatic Control*, Vol. AC-28, No. 1, pp. 162-71, (1983).

[Lathrop 1985].
LATHROP, R. H., "Parallelism in Manipulator Dynamics ", *Int. J. Robotics Research*, Vol. 4, No. 2, pp. 80-102, (1985).

[Lee and Chung 1984].
LEE, C. S. G. AND CHUNG, M. J., "An Adaptive Control Strategy for Mechanical Manipulators ", *IEEE Trans. Automatic Control*, Vol. AC-29, No. 9, pp. 837-40, (1984).

[Lee and Chang 1986].
LEE, C. S. G. AND CHANG, P. R., "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation ", *IEEE Trans. Syst., Man, Cyber.*, Vol. SMC-16, No. 4, pp. 532-542, (1986).

[Lippmann 1987].
LIPPMANN, R. P., "An Introduction to Computing with Neural Nets ", *IEEE ASSP Magazine*, pp. 4-22, (1987).

[Luh and Lin 1982].
LUH, J. Y. S. AND LIN, C. S., "Scheduling of Parallel Computation for a Computer Controlled Mechanical Manipulator ", *IEEE Trans. Syst., Man, Cyber.*, Vol. SMC-12, No. 2, pp. 214-234, (1982).

[McClelland and Rumelhart 1988].
MCCLELLAND, J. L. AND RUMELHART, D. E., *Explorations in Parallel Distributed Processing*, (MIT Press, 1988).

[Miche 1986].
MICHE, D., *On Machine Intelligence*, p. 92, (Ellis Horwood, 1986).

[Orin 1984].
ORIN, D. E., "Pipelined Approach to Inverse Plant Plus Jacobian Control of Robot Manipulators", In *Proc. IEEE Int. Conf. Robotics and Automation*, (1984) , pp. 169-175.

[Pao and Sobajic 1987].
PAO, Y. H. AND SOBAJIC, D. J., "Artificial Neural-Net Based Intelligent Robotics Control", In *Proc. SPIE Robotics/IECON 87 Meeting*, Vol. 848, (1987) , pp. 542-9.

[Paul,Rong and Zhang 1983].
PAUL, R. P., RONG, M., AND ZHANG, H., "The Dynamics of the PUMA Manipulator", In *Proc. American Control Conf.*, (1983) .

[Psaltis,Sideris and Yamamura 1987].
PSALTIS, D., SIDERIS, A., AND YAMAMURA, A., "Neural Controllers", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 4, (1987) , pp. 551-8.

[Psaltis,Sideris and Yamamura 1988].
PSALTIS, D., SIDERIS, A., AND YAMAMURA, A. A., "A Multilayered Neural Network Controller ", *IEEE Control Sestems Magazine*, pp. 17-21, (1988).

[Ritter,Martinetz and Schulten 1989].
RITTER, H. J., MARTINETZ, T. M., AND SCHULTEN, K. J., "Topology-Conserving Maps for Learning Visuo-Motor-Coordination ", *Neural Networks*, Vol. 2, pp. 159-68, (1989).

[Roth 1988].
ROTH, M. W., "Neural-Network Technology and its Applications ", *Johns Hopkins APL Technical Digest*, Vol. 9, No. 3, pp. 242-53, (1988).

[Rumelhart and McClelland 1986].
RUMELHART, D. E. AND MCCLELLAND, J. L., *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, Vol. 1,2, (MIT Press, 1986).

[Rumelhart,Hinton and Williams 1986a].
RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J., "Learning Representations by Back-propagating Errors ", *Nature*, Vol. 323, pp. 533-6, (1986).

[Sobajic,Lu and Pao 1988].
SOBAJIC, D. J., LU, J. J., AND PAO, Y. H., "Intelligent Control of the Intelledex 605T Robot Manipulator", In *Proc. IEEE Int. Conf. Neural Networks*, Vol. 2, (1988) , pp. 633-40.

[Swartz 1984].
SWARTZ, N. M., "Arm Dynamics Simulation ", *J. Robotics Systems*, Vol. 1, No. 1, pp. 83-100, (1984).

[Tang and Tourassis 1989].
TANG, K.-M. W. AND TOURASSIS, V. D., "Mathematical Deficiencies of Numerically Simplified Dynamic Robot Models ", *IEEE Trans. Automatic Control*, Vol. AC-34, No. 10, pp. 1109-11, (1989).

[Tarn,Bejczy,Han and Yun 1985].
TARN, T. J. TA A. K. BEJCZY, HAN, S., AND YUN, X. P., "Inertia Parameters of PUMA 560 Robot Arm", Robotics Laboratory Report SSM-RL-85-01, (Dept. Systems Science and Mathematics, Washington Univ.,1985) .

[Tarn,Bejczy,Yun and Ding 1986].
TARN, T. J., BEJCZY, A. K., YUN, X., AND DING, X., "Dynamic Equations for Six-Link PUMA 560 Robot Arm", Robotics Laboratory Report SSM-RL-86-05, (Dept. Systems Science and Mathematics, Washington Univ.,1986) .

[Walker and Orin 1982].
WALKER, M. W. AND ORIN, D. E., "Efficient Dynamic Computer Simulation of Robotic Mechanisms ", *Trans. ASME, J. Dyn. Syst., Meas. and Cntrl.*, Vol. 104, pp. 205-11, (1982).

[Zalzala and Morris 1988].
ZALZALA, A. M. S. AND MORRIS, A. S., "An Optimum Trajectory Planner for Robot Manipulators in Joint-Space and Under Physical Constraints", Research Report #349, (Dept. Control Eng.,Univ. of Sheffield, UK,1988) .

[Zalzala and Morris 1989a].
ZALZALA, A. M. S. AND MORRIS, A. S., "A Distributed Pipelined Architecture of the Recursive Lagrangian Equations of Motion for Robot Manipulators with VLSI Implementation", Research Report #353, (Dept. Control Eng.,Univ. of Sheffield, UK,1989) .

[Zalzala and Morris 1989b].
ZALZALA, A. M. S. AND MORRIS, A. S., "An On-Line Distributed Minimum-Time Trajectory Generator for Intelligent Robot Manipulators", Research Report #358, (Dept. Control Eng.,Univ. of Sheffield, UK,1989) .

[Zalzala and Morris 1989g].
ZALZALA, A. M. S. AND MORRIS, A. S., "A Fast Trajectory Tracker for Intelligent Robot Manipulators", In *Proc. Int. Conf. Applications of Transputers*, (University of Liverpool,Liverpool,UK,(to appear),1989) .

[Zalzala and Morris 1989f].
ZALZALA, A. M. S. AND MORRIS, A. S., "Structured Motion Planning in The Local Configuration Space ", *Robotica*, (accepted for publication), (1989).

Appendix (A) :

Nodes of the Acceleration-Related Term (Hidden Layer #1)

```
void AccNodes()
{
float   S2,S3,S23,SS23,S22,C2,C3,C23,C22,SS2,SS3,CC3,S33,CC2,CC23,C33,S4,SS4,S44,
        C4,CC4,C44,S5,SS5,S55,C5,CC5,C55,S6,SS6,S66,C6,CC6,C66,S46,S233,S2233,
        C2233,C235,S223,C223;
switch (node)
{
case 1 : /*------------ NODE #1 == Inertia Matrix Element : D11 ----------------*/
        S2 = sin(values[1]);
        S3 = sin(values[2]);
        S4 = sin(values[3]);
        S5 = sin(values[4]);
        S6 = sin(values[5]);
        SS2 = S2*S2;
        SS3 = S3*S3;
        SS4 = S4*S4;
        SS5 = S5*S5;
        SS6 = S6*S6;
        S22 = sin(2.*values[1]);
        S33 = sin(2.*values[2]);
        S44 = sin(2.*values[3]);
        S55 = sin(2.*values[4]);
        S66 = sin(2.*values[5]);
        S23 = sin(values[1]+values[2]);
        SS23 = S23*S23;
        S233 = sin(values[1]+values[2]+values[2]);
        C2 = cos(values[1]);
        C3 = cos(values[2]);
        C4 = cos(values[3]);
        C5 = cos(values[4]);
        C6 = cos(values[5]);
        CC2 = C2*C2;
        CC3 = C3*C3;
        CC4 = C4*C4;
        CC5 = C5*C5;
        CC6 = C6*C6;
        C23 = cos(values[1]+values[2]);
        CC23 = C23*C23;
        C235 = cos(values[1]+values[2]+values[4]);
activation[1][node] = rm1*ryy1+rm2*(rxx2*SS2+ryy2*CC2+raa2*CC2+2.*ra2*CC2*rx2)+
        rm3*(rxx3*SS23+rzz3*CC23+rdd3*rdd3+ raa2*C2*C2+raa3*CC23+2.*ra2*ra3*
        C2*C23+2.*rx3*(ra2*C2*C23+ra3*CC23)+2.*ry3*rdd3+2.*rz3*(ra3*C23*S23+
        ra2*C2*S23))+rm4*(rxx4*CC4*(SS2+S3*S3+2.*S2*S3*C23)+ryy4*(CC2-2.*S2*S3*
        C23-SS3)+rzz4*SS4*(SS2+SS3+2.*S2*S3*C23)-2.*ry4*S23*(C2*ra2+S23*rdd4+C23*
        ra3)-2*rz4*(ra3*S4*(S2*S233-CC3)-C2*S4*C23*ra2-S4*C23*S23*rdd4+C4*rdd3)+
        rdd3*rdd3+CC2*raa2+SS23*rdd4*rdd4+(1-SS23)*raa3+2.*C2*C23*ra2*ra3+2.*C2
        *S23*ra2*rdd4+2.*C23*S23*ra3*rdd4);
activation[1][node] = activation[1][node]+rm5*(rxx5*(SS23*(1-SS4*CC5)-(SS2-CC2)
        *(SS5+2.*C3*S3*C4*C5*S5)-2.*(SS3-CC3)*C2*S2*C4*C5*S5-2.*SS5*(SS3+2.*S2
        *S3*C23))+ryy5*SS4*(S2*C3*S23+S3*S3*CC2)+rzz5*(CC23+(SS2-CC2)*(SS5+2.*C3
        *S3*C4*C5*S5)+2*(S3*S3-CC3)*C2*S2*C4*C5*S5+SS5*(2.*SS3+4*S2*S3*C23-SS4*
        SS23))+2.*(((S2*S23-C3)*ra2-CC23*ra3-(C2*S2+C3*S3-2.*S2*S3*S23)*rdd4)*S4+
        C4*rdd3)*ry5+2.*((S2*C235+S2*S5*S23*(1-C4)+C3*C4*S5+S3*C5)*ra2+(C4*S5+CC23
        +(C2*S2+C3*S3-2.*S2*S3*S23)*C5)*ra3+(C4*S5*(C2*S2+C3*S3-2.*S2*S3*S23)
        +C5*(SS2+SS3+2.*S2*S3*C23))*rdd4+S4*S5*rdd3)*rz5+rdd3*rdd3+CC2*raa2+
        CC23*raa3+(1-CC23)*rdd4*rdd4+2.*C2*S23*ra2*rdd4+2.*(C3-S2*S23)*ra2*
        ra3+2.*(C2*S2+C3*S3-2.*S2*S3*S23)*ra3*rdd4);
activation[1][node] = activation[1][node]+rm6*(rxx6*(-SS23*(SS4-2.*SS4*SS6+CC4*SS5
        *CC6-CC5*CC6+0.5*S44*C5*S66)+SS5*CC6+(C4*S55*CC6-S4*S5*S66)*(0.5*S22+0.5*
        S33-2*S2*S3*S23))+ryy6*(SS23*(SS4-2.*SS4*SS6+CC5*SS6+SS4*SS5*SS6+0.5*S44*
        C5*S66)+SS5*S6*S6+(S4*S5*S66+C4*S55*SS6)*(0.5*S22+0.5*S33-2.*S2*S3*S23))+
        rzz6*(CC23*CC5+CC4*SS5*SS23-0.5*S22*C4*S55+0.5*S33*C4*S55-S2*S3*S23*
        C4*S55)+2.*rz6*(ra2*(C3*C4*S5-S2*S23*C4*C5+S3*C5+S2*C23*C5)+ra3*(CC23*
        C4*S5+0.5*S22*C5+0.5*S33*C5-2.*S2*S3*S23*C5)+S4*S5*rdd3+rdd4*(0.5*
        S22*C4*S5+0.5*S33*C4*S5-2.*S2*S3*S23*C4*S5+C5*SS23))+raa2*CC2+raa3*
        CC23+rdd3*rdd3+rdd4*rdd4*SS23+2.*ra2*ra3*C3-2.*ra2*ra3*S2*S23+2.*ra2*
        rdd4*S2*C23+2.*ra2*rdd4*S3+ra3*rdd4*S22+ra3*rdd4*S33-4*ra3*rdd4*S2*
        S3*S23);
activation[1][node] = activation[1][node]+rm6*((C6*rx6-S6*ry6)*(-S22*S33*C4*C5*
        ra3-S22*S33*S5*rdd4-S22*C3*S5*ra2+2.*S22*SS3*C4*C5*rdd4+2.*S22*SS3*S5*
        ra3-S22*S3*C4* C5*ra2-S22*C4*C5*rdd4-S22*S5*ra3-2.*SS2*S33*C4*C5*rdd4
        +2.*SS2*S33*S5*ra3-2.*SS2*C3*C4*C5*ra2+4*SS2*SS3*C4*C5*ra3+4*SS2*SS3*S5
        *rdd4+2.*SS2*S3*S5*ra2-2.*SS2*C4*C5*ra3-2.*SS2*S5*rdd4+S33*C4*C5*rdd4
        -S33*S5*ra3+2.*C3*C4*C5*ra2-2.*SS3*C4*C5*ra3-2.*SS3*S5*rdd4-2.*S3*S5*
        ra2+2.*C4*C5*ra3+2.*S4*C5*rdd3)+(C6*ry6+S6*rx6)*(S22*S33*S4*ra3+2.*S22*
        S3*S4*rdd4+S22*S3*S4*ra2-S22*S4*rdd4+2.*SS2*S33*S4*rdd4+2.*SS2*C3*S4*
        ra2-4*SS2*SS3*S4*ra3+2.*SS2*S4*ra3-S33*S4*rdd4-2.*C3*S4*ra2+2.*SS3*S4
        *ra3+2.*C4*rdd3-2*S4*ra3));
break; /*--------------------------------------------------------------*/

case 2 : /*------------ NODE #2 == Inertia Matrix Element : D12 ----------------*/
        S2 = sin(values[1]);
        S3 = sin(values[2]);
        S4 = sin(values[3]);
        S5 = sin(values[4]);
        S6 = sin(values[5]);
        SS5 = S5*S5;
        SS6 = S6*S6;
        S44 = sin(2.*values[3]);
        S55 = sin(2.*values[4]);
        S66 = sin(2.*values[5]);
        S23 = sin(values[1]+values[2]);
        C2 = cos(values[1]);
        C3 = cos(values[2]);
        C4 = cos(values[3]);
        C5 = cos(values[4]);
        C6 = cos(values[5]);
        C44 = cos(2.*values[3]);
        C66 = cos(2.*values[5]);
        C23 = cos(values[1]+values[2]);
activation[1][node]= rm2*ra2*rz2*S2+rm3*((rdd3*rx3+ra3*rdd3+ra3*ry3)*S23-rdd3
        *rz3*C23+ra2*(rdd3+ry3)*S2)+rm4*((rdd4*ry4-rdd3*rdd4-rdd4*rz4*C4)*C23+
        (ra3*rz4*C4-rdd3*rz4*S4+ra3*rdd4)*S23+ra2*(rdd3+rz4*C4)*S2-(rxx4-
        rzz4)*S23*C4*S4)+rm5*(0.5*(rxx5-rzz5)*(S23*S44*S5*S5-C23*S4*S55)-
        0.5*(rxx5-ryy5)*S23*S44-(rdd3*rdd4+rdd4*ry5*C4+rdd4*rz5*S4*S5+rdd3*
        rz5*C5)*C23+((rdd3*rz5*S5+ra3*ry5)*C4+(ra3*rz5*S5-rdd3*ry5)*S4+ra3*
        rdd3)*S23+ra2*S2*(rdd3+ry5*C4+rz5*S4*S5))+rm6*(0.5*(rxx6-ryy6)*
        ((S4*S55*SS6-C4*C5*S66)*C23-(C44*C5*S66+S44*(C66+SS5*SS6)))+0.5*
        (rxx6-rzz6)*(S23*S44*S5*S5-C23*S4*S55)-(rdd4*rz6*S4*S5+rdd3*rz6*C5+
        rdd3*rdd4)*C23+((rdd3*C4+ra3*S4)*C5*rz6+ra3*rdd3)*S23+ra2*S2*(rdd3+
        rz6*S4*S5));
activation[1][node] = activation[1][node]+rm6*((ry6*C6+rx6*S6)*(-rdd4*C2*C3*C4+
        ra3*C2*S3*C4-rdd3*C2*S3*S4+ra3*S2*C3*C4-rdd3*S2*C3*S4+rdd4*S2*S3*C4+
        ra2*S2*C4)+(ry6*S6-rx6*C6)*(rdd4*C2*C3*S4*C5-rdd3*C2*C3*S5-rdd3*C2*S3
        *C4*C5-ra3*C2*S3*S4*C5-rdd3*S2*C3*C4*C5-ra3*S2*C3*S4*C5-rdd4*S2*S3*S4
        *C5+rdd3*S2*S3*S5-ra2*S2*S4*C5));
break; /*--------------------------------------------------------------*/

case 3 : /*------------ NODE #3 == Inertia Matrix Element : D13 ----------------*/
        S4 = sin(values[3]);
        S5 = sin(values[4]);
        S6 = sin(values[5]);
        SS5 = S5*S5;
        SS6 = S6*S6;
        S44 = sin(2.*values[3]) ;
        S55 = sin(2.*values[4]);
        S66 = sin(2.*values[5]);
        S23 = sin(values[1]+values[2]);
        C4 = cos(values[3]);
        C5 = cos(values[4]);
        C6 = cos(values[5]);
        C44 = cos(2.*values[3]);
        C66 = cos(2.*values[5]);
        C23 = cos(values[1]+values[2]);
activation[1][node]= rm3*((rdd3*rx3+ra3*rdd3+ra3*ry3)*S23-rdd3*rz3*C23)+rm4*
        ((0.5*(rzz4-rxx4)*S44+ra3*rz4*C4-rdd3*rz4*S4+ra3*rdd3)*S23-(rdd3*rdd4
        -rdd3*ry4+rdd4*rz4*C4)*C23)+rm5*((0.5*(rzz5-rxx5)*S4*S55-rdd4*ry5*C4-
        rdd4*rz5*S4*S5-rdd3*rz5*C5-rdd3*rdd4)*C23+(0.5*((rxx5-ryy5)*SS5-rxx5
        +ryy5)*S44+(rdd3*rz5*S5+ra3*ry5)*C4+(ra3*rz5*S5-rdd3*ry5)*S4+ra3*rdd3)*
        S23)+rm6*((0.5*((rxx6-ryy6)*SS6-(rxx6-rzz6))*S4*S55-(0.5*(rxx6-
        ryy6)*C4*S66+rdd4*rz6*S4)*S5-rdd3*(rdd4+rz6*C5))*C23+(0.5*((rxx6-
        rzz6)*SS5-(rxx6-ryy6)*(C66+SS5*SS6))*S44-0.5*(rxx6-ryy6)*C44*C5*
        S66+(rdd3*C4+ra3*S4)*S5*rz6+ra3*rdd3)*S23)

        +rm6*((C6*ry6+S6*rx6)*(S23*C4*ra3-C23*C4*rdd4-S23*S4*rdd3)+(S6*ry6-rx6*
        C6)*(C23*S4*C5*rdd4-C23*S5*rdd3-S23*C4*C5*rdd3-S23*S4*C5*ra3));
break; /*-------------------------------------------------------------------*/

case 4 : /*------------ NODE #4 == Inertia Matrix Element : D14 ----------------*/
        S4 = sin(values[3]);
        S5 = sin(values[4]);
        S6 = sin(values[5]);
        SS5 = S5*S5;
        SS6 = S6*S6;
        S55 = sin(2.*values[4]);
        S66 = sin(2.*values[5]);
        S23 = sin(values[1]+values[2]);
        C2 = cos(values[1]);
        C4 = cos(values[3]);
        C5 = cos(values[4]);
        C6 = cos(values[5]);
        C23 = cos(values[1]+values[2]);
activation[1][node]= rm4*((ryy4+rz4*(rdd4*C4-ra3*S4))*C23-rz4*(rdd4*S4*S23-
        ra2*C2*C4))+rm5*((rxx5-rzz5)*(C23*SS5+0.5*S23*S4*S55)+((ra3*rz5*S5+
        rdd3*ry5)*C4+(rdd3*rz5*S5-ra3*ry5)*S4+rzz5)*C23+rdd4*(rz5*C4*S5-ry5*
        S4)*S23+ra2*rz5*C2*C4*S5)+rm6*((rz6*S5*(ra3*C4+rdd3*S4)+rzz6+(rxx6-
        rzz6)*SS5-(rxx6-ryy6)*SS6)*C23+0.5*(2.*rdd4*rz6*C4*S5+(rxx6-rzz6-SS6*
        (rxx6-ryy6))*C4*S55-(rxx6-ryy6)*S4*S5*S66)*S23+ra2*rz6*C2*C4*S5)

        +rm6*((S6*rx6+C6*ry6)*(C23*C4*rdd3-C23*S4*ra3-S23*S4*rdd4-C2*S4*ra2)
        +(rx6*C6-S6*ry6)*(C23*C4*C5*ra3-C23*S4*C5*rdd3+C2*C4*C5*ra2+S23*C4*C5*
        rdd4-S23*C4*C5*ra3));
break; /*-------------------------------------------------------------------*/

case 5 : /*------------ NODE #5 == Inertia Matrix Element : D15 ----------------*/
        S4 = sin(values[3]);
        S5 = sin(values[4]);
        S6 = sin(values[5]);
        S23 = sin(values[1]+values[2]);
        SS6 = S6*S6;
        S66 = sin(2.*values[5]);
        C2 = cos(values[1]);
        C4 = cos(values[3]);
        C5 = cos(values[4]);
        C6 = cos(values[5]);
        C23 = cos(values[1]+values[2]);
activation[1][node]= rm5*(((ryy5+rdd4*rz5*C5)*S4+rdd3*rz5*S5)*S23+rz5*(ra3*S4
        -rdd3*C4)*C23*C5+ra2*rz5*C2*S4*S5)+rm6*((rxx6-ryy6)*(S23*S4*SS6-0.5*
        S66*(C23*S5+C4*C5))+(ryy6*S4+rz6*C5*(ra3+rdd4)+rdd3*rz6*S5)*S23+rz6*
        (ra3*S4-rdd3*C4)*C23*C5)

        +rm6*((rx6*C6-S6*ry6)*( C23*C4*S5*rdd3-C23*S4*S5*ra3-S23*S4*S5*rdd4+
        S23*C5*rdd3-C2*S4*S5*ra2));
break; /*-------------------------------------------------------------------*/
```

```
case 6 : /*------------ NODE #6 == Inertia Matrix Element : D16 ----------------*/
        S4 = sin(values[3]);
        S5 = sin(values[4]);
        S6 = sin(values[5]);
        S23 = sin(values[1]+values[2]);
        C2 = cos(values[1]);
        C4 = cos(values[3]);
        C5 = cos(values[4]);
        C6 = cos(values[5]);
        C23 = cos(values[1]+values[2]);
activation[1][node]= rm6*rzz6*(C23*C5- S23*C4*C5)

        +rm6*((C6*ry6+S6*rx6)*(C23*C4*C5*rdd3-C23*S4*C5*ra3-S23*S4*C5*rdd4-S23*
        S5*rdd3-C2*S4*C5*ra2)+(rx6*C6-S6*ry6)*(S23*C4*ra3+C23*S4*rdd3+S23*C4*
        rdd4+C2*C4*ra2));
break; /*-------------------------------------------------------------------*/

case 7 : /*------------ NODE #7 == Inertia Matrix Element : D22 ----------------*/
        S3 = sin(values[1]);
        S4 = sin(values[2]);
        S5 = sin(values[3]);
        S6 = sin(values[4]);
        SS4 = S4*S4;
        SS5 = S5*S5;
        S44 = sin(2.*values[2]) ;
        S66 = sin(2.*values[4]);
        C3 = cos(values[1]);
        C4 = cos(values[2]);
        C5 = cos(values[3]);
        C6 = cos(values[4]);
        CC3 = C3*C3;
        CC4 = C4*C4;
        CC6 = C6*C6;
activation[1][node]= rm2*(rzz2+raa2+2.*ra2*rx2)+rm3*(ryy3+raa2+2.*raa3*rx3+2.*ra2
        *(rx3+ra3)*C3+2.*ra2*rz3*S3)+rm4*((rxx4-rzz4)*S4*S4-2.*rz4*S4*(ra2*C3+
        ra3)-2.*ry4*(ra2*S3+rdd4)+raa2+raa3+rdd4*rdd4+rzz4+2.*ra2*(ra3*C3+rdd4*
        S3))+rm5*(2.*(ra2*C3+ra3)*((C4*S5+C5)*rz5-S4*ry5)+2.*ra2*(ra3*C3+rdd4
        *S3)+SS4*(SS5*(rzz5-rxx5)-ryy5+rxx5)+raa2+raa3+rdd4*rdd4+ryy5)+rm6*(2.*
        rz6*(ra2*C3*C4*S5+ra2*S3*C5+ra3*C4*C5+rdd4*C5)+2.*ra2*(ra3*C3+rdd4*
        S3)+(ryy6-rxx6)*(S6*S6*(CC4*SS5-1)-SS4* CC6-0.5*S44*C5*S66)+(rzz6-
        rxx6)*SS4*SS5+raa2+raa3+rdd4*rdd4+ryy6)

        +rm6*((C6*rx6-S6*ry6)*(2.*C3*C4*C5*ra2-2.*S3*S5*ra2+2.*C4*C5*ra3-2.
        *S5*rdd4)-2.*(C6*ry6+S6*rx6)*(C3*S4*ra2+S4*ra3));
break; /*-------------------------------------------------------------------*/

case 8 : /*------------ NODE #8 == Inertia Matrix Element : D23 ----------------*/
        S3 = sin(values[1]);
        S4 = sin(values[2]);
        S5 = sin(values[3]);
        S6 = sin(values[4]);
        SS4 = S4*S4;
        SS5 = S5*S5;
        SS6 = S6*S6;
        S44 = sin(2.*values[2]);
        S66 = sin(2.*values[4]);
        C3 = cos(values[1]);
        C4 = cos(values[2]);
        C5 = cos(values[3]);
        C6 = cos(values[4]);
        CC3 = C3*C3;
        CC5 = C5*C5;
        CC6 = C6*C6;
        C66 = cos(2.*values[4]);
activation[1][node]= rm3*((ra3+rx3)*(ra2*C3+ra2*rz3*S3+ra3*(2.*rx3+ra3))+ryy3)
```

```
        +rm4*((ra3-S4*rz4)*ra2*C3+(rdd4-ry4)*ra2*S3+(rxx4-rzz4)*SS4+ra3*(ra3-
        2*S4*rz4)+rdd4*(rdd4-2.*ry4)+rzz4)+rm5*((rxx5*CC5+rzz5*SS5)*S4*S4+ryy5*
        CC4+ra2*S3*(rdd4+rz5*C5)+ra2*C3*(rz5*C4*S5-ry5*S4+ra3)+2.*rdd4*rz5*C5+
        rdd4*rdd4+2.*ra3*(rz5*C4*S5-ry5*S4)+raa3)+rm6*((ra2*C3+2.*ra3)*rz6*
        C4*S5+(ra2*S3+2.*rdd4)*rz6*C5+ra2*(ra3*C3+rdd4*S3)+raa3+rdd4*rdd4-
        0.5*(rxx6-ryy6)*S44*C5*S66+SS4*(C66*(rxx6-ryy6)-SS5*(CC6*rxx6+S6*
        S6*ryy6-rzz6))+SS6*(rxx6-ryy6)+ryy6)

        +rm6*((C6*rx6-S6*ry6)*(C3*C4*C5*ra2-S3*S5*ra2+2.*C4*C5*ra3-2.*S5*rdd4)
        -(C6*ry6+S6*rx6)*(C3*S4*ra2-2.*S4*ra3));
break; /*------------------------------------------------------------------*/

case 9 : /*------------ NODE #9 == Inertia Matrix Element : D24 ----------------*/
        S3 = sin(values[1]);
        S4 = sin(values[2]);
        S5 = sin(values[3]);
        S6 = sin(values[4]);
        SS6 = S6*S6;
        S66 = sin(2.*values[4]);
        C4 = cos(values[2]);
        C5 = cos(values[3]);
        C6 = cos(values[4]);
activation[1][node]= -rm4*rz4*C4*(rdd4+ra2*S3)+rm5*((((rzz5-rxx5)*C5-rz5*(ra2*
        S3+rdd4))*S4*S5-ry5*C4*(ra2*S3+rdd4))+rm6*(S5*(S4*(C5*(rzz6-C6*C6*rxx6
        -SS6*ryy6)-rz6*(ra2*S3+rdd4))-0.5*(rxx6-ryy6)*C4*S66))

        +rm6*((S6*ry6-C6*rx6)*(S3*S4*C5*ra2+S4*C5*rdd4)-(C6*ry6+S6*rx6)*(S3*
        C4*ra2-C4*rdd4));
break; /*------------------------------------------------------------------*/

case 10 : /*------------ NODE #10 == Inertia Matrix Element : D25 --------------*/
        S3 = sin(values[1]);
        S4 = sin(values[2]);
        S5 = sin(values[3]);
        S6 = sin(values[4]);
        SS6 = S6*S6;
        S66 = sin(2.*values[4]);
        C3 = cos(values[1]);
        C4 = cos(values[2]);
        C5 = cos(values[3]);
        C6 = cos(values[4]);
activation[1][node]= rm5*(C4*(ryy5+rz5*C5*(ra2*S3+rdd4))+rz5*S5*(ra2*C3+ra3))
        +rm6*(C4*(SS6*(rxx6-ryy6)+ryy6+rz6*C5*(ra2*S3+rdd4))+rz6*S5*(ra2*C3+
        ra3)+0.5*(rxx6-ryy6)*S4*C5*S66)

        +rm6*((C6*rx6-S6*ry6)*(C3*C5*ra2+C5*ra3-S3*C4*S5*ra2-C4*S5*rdd4));
break; /*------------------------------------------------------------------*/

case 11 : /*------------ NODE #11 == Inertia Matrix Element : D26 --------------*/
        S3 = sin(values[1]);
        S4 = sin(values[2]);
        S5 = sin(values[3]);
        S6 = sin(values[4]);
        C3 = cos(values[1]);
        C4 = cos(values[2]);
        C5 = cos(values[3]);
        C6 = cos(values[4]);
activation[1][node]= rm6*rzz6*S4*S5

        +rm6*((S6*ry6-C6*rx6)*(S3*S4*ra2+S4*rdd4)-(C6*ry6+S6*rx6)*(C3*S5*ra2+
        S3*C4*C5*ra2+C4*C5*rdd4+S5*ra3));
break; /*------------------------------------------------------------------*/

case 12 : /*------------ NODE #12 == Inertia Matrix Element : D33 --------------*/
        S4 = sin(values[1]);
        S5 = sin(values[2]);
        S6 = sin(values[3]);
        SS4 = S4*S4;
        SS5 = S5*S5;
        SS6 = S6*S6;
        S44 = sin(2.*values[1]);
        S66 = sin(2.*values[3]);
        C4 = cos(values[1]);
        C5 = cos(values[2]);
        C6 = cos(values[3]);
        C44 = cos(2.*values[1]);
activation[1][node]= rm3*(2.*ra3*rx3+raa3+ryy3)+rm4*((rxx4-rzz4)*SS4-2.*ra3*rz4*
        S4+raa3+rdd4*rdd4-2.*rdd4*ry4+rzz4)+rm5*(SS4*(SS5*(rzz5-rxx5)+rxx5-
        ryy5)+2.*rz5*(ra3*C4*C5+rdd4*C5)-2.*ra3*ry5*S4+raa3+rdd4*rdd4+ryy5)+
        rm6*((rxx6-ryy6)*(S44*C5*S66+S4*S4*(1+SS5*SS6)+C44*SS6)-(rxx6-rzz6)*
        S4*S4*SS5+2.*rz6*(ra2*C4*S5+rdd4*C5)+raa3+rdd4*rdd4+ryy6)

        + rm6*(-2.*S4*ra3*(C6*ry6+S6*rx6)+2.*(C6*rx6-S6*ry6)*(C4*C5*ra3-S5*
        rdd4));
break; /*---------------------------------------------------------------------*/

case 13 : /*------------ NODE #13 == Inertia Matrix Element : D34 --------------*/
        S4 = sin(values[1]);
        S5 = sin(values[2]);
        S6 = sin(values[3]);
        SS6 = S6*S6;
        S66 = sin(2.*values[3]);
        C4 = cos(values[1]);
        C5 = cos(values[2]);
        C6 = cos(values[3]);
activation[1][node]= -rm4*rdd4*rz4*C4+rm5*(S4*S5*(C5*(rzz5-rxx5)-rdd4*rz5)
        -rdd4*ry5*C4)+rm6*(S5*(S4* C5*(rzz6-C6*C6*rxx6-SS6*ryy6)-0.5*(rxx6-
        ryy6)*C4*S66-rdd4*rz6*S4))

        +rm6*(S4*C5*rdd4*(S6*ry6-C6*rx6)-C4*rdd4*(S6*rx6+C6*ry6));
break; /*---------------------------------------------------------------------*/

case 14 : /*------------ NODE #14 == Inertia Matrix Element : D35 --------------*/
        S4 = sin(values[1]);
        S5 = sin(values[2]);
        S6 = sin(values[3]);
        SS6 = S6*S6;
        S66 = sin(2.*values[3]);
        C4 = cos(values[1]);
        C5 = cos(values[2]);
        C6 = cos(values[3]);
activation[1][node]= rm5*(C4*(rdd4*rz5*C5+ryy5)+ra3*rz5*S5)+rm6*(C4*(rdd4*rz6
        *C5+SS6*(rxx6-ryy6)+ryy6)+0.5*(rxx6-ryy6)*S4*C5*S66+ra3*rz6*S5)

        +rm6*((S6*ry6-C6*rx6)*(C4*S5*rdd4-C5*ra3));
break; /*---------------------------------------------------------------------*/

case 15 : /*------------ NODE #15 == Inertia Matrix Element : D36 --------------*/
        S4 = sin(values[1]);
        S5 = sin(values[2]);
        S6 = sin(values[3]);
        C4 = cos(values[1]);
        C6 = cos(values[3]);
activation[1][node]= rm6*rzz6*S4*S5

        +rm6*(S4*rdd4*(S6*ry6-C6*rx6)-(C6*ry6+S6*rx6)*(C4*C5*rdd4+S5*ra3));
break; /*---------------------------------------------------------------------*/

case 16 : /*------------ NODE #16 == Inertia Matrix Element : D44 --------------*/
        S5 = sin(values[1]);
        S6 = sin(values[2]);
        SS5 = S5*S5;
        SS6 = S6*S6;
        C5 = cos(values[1]);
        CC5 = C5*C5;
activation[1][node]= rm4*ryy4+rm5*(rxx5*SS5+rzz5*CC5)+rm6*(SS5*(rxx6+SS6*
        (ryy6-rxx6))+CC5*rzz6);
break; /*----------------------------------------------------------------------*/

case 17 : /*------------ NODE #17 == Inertia Matrix Element : D45 --------------*/
        S5 = sin(values[1]);
        S6 = sin(values[2]);
        S66 = sin(2.*values[2]);
activation[1][node]= 0.5*rm6*S5*S66*(ryy6-rxx6);
break; /*----------------------------------------------------------------------*/

case 18 : /*------------ NODE #18 == Inertia Matrix Element : D46 --------------*/
        C5 = cos(values[1]);
activation[1][node]= rm6*rzz6*C5;
break; /*----------------------------------------------------------------------*/

case 19 : /*------------ NODE #19 == Inertia Matrix Element : D55 --------------*/
        S6 = sin(values[1]);
        SS6 = S6*S6;
        C6 = cos(values[1]);
        CC6 = C6*C6;
activation[1][node]= rm5*ryy5+rm6*(SS6*rxx6+CC6*ryy6);
break; /*----------------------------------------------------------------------*/

case 20 : /*------------ NODE #20 == Inertia Matrix Element : D66 --------------*/
activation[1][node]= rm6*rzz6;
break; /*----------------------------------------------------------------------*/

}
}
```

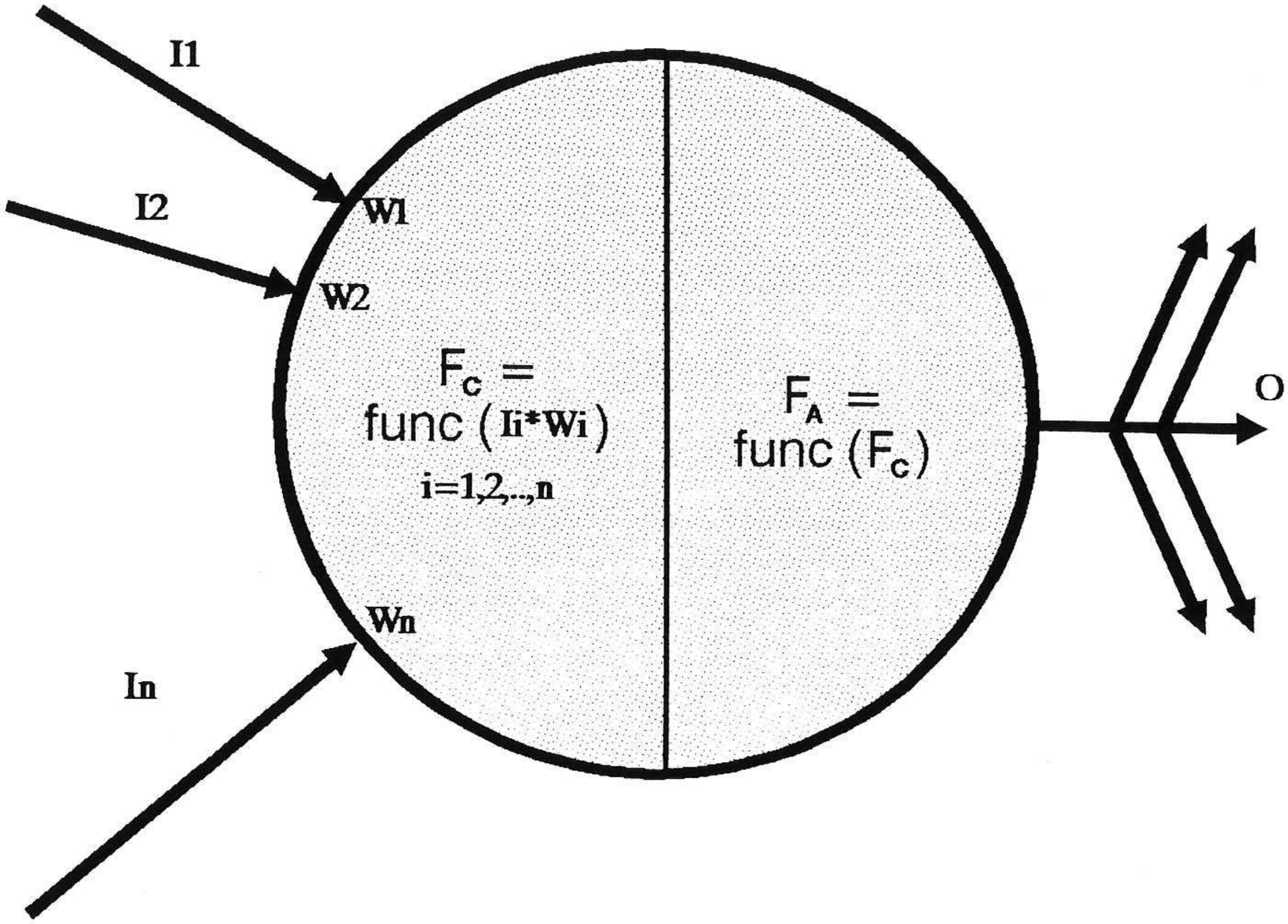

Figure (1) : The Neuron

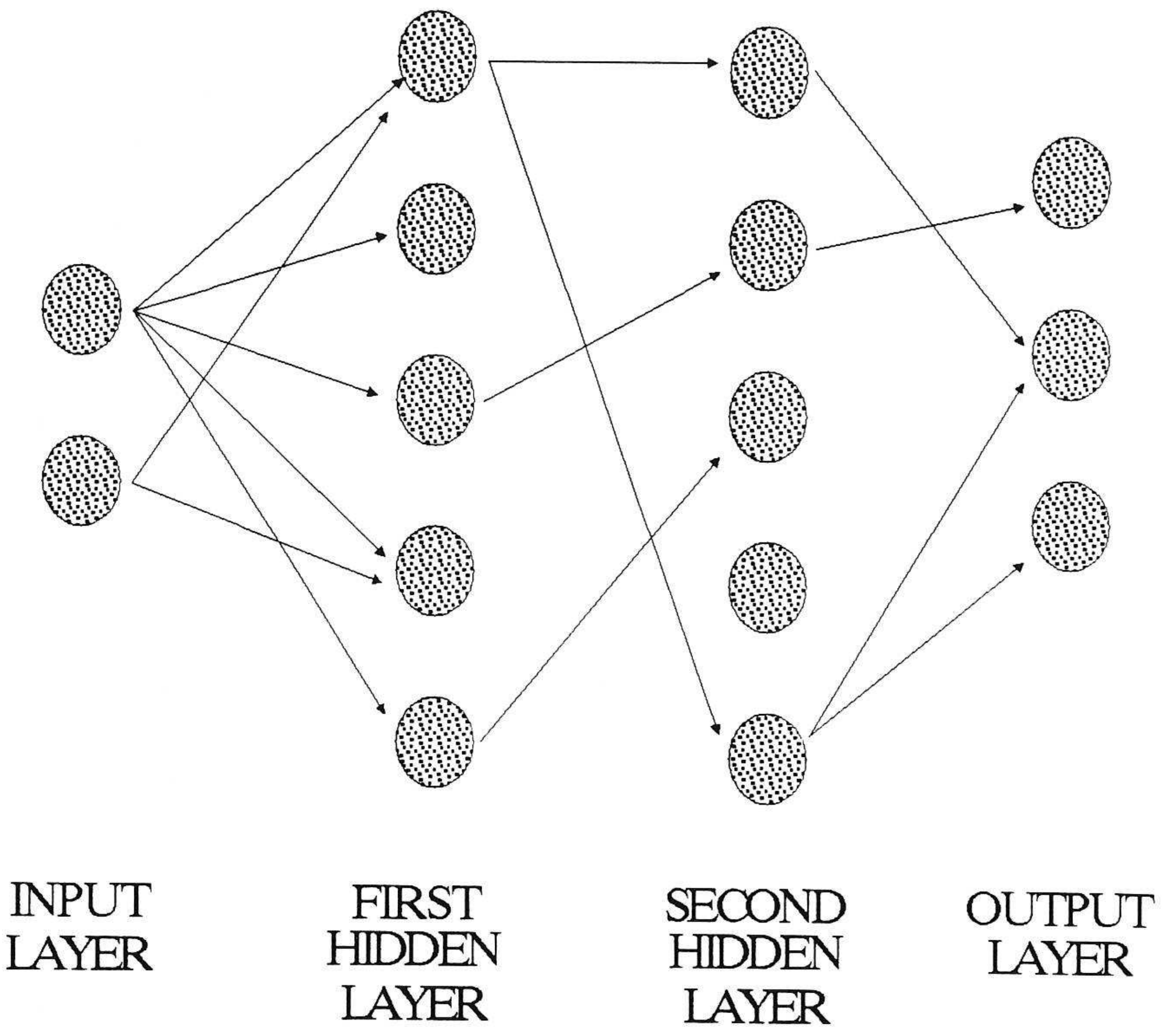

Figure (2) : A Multi-Layered Network

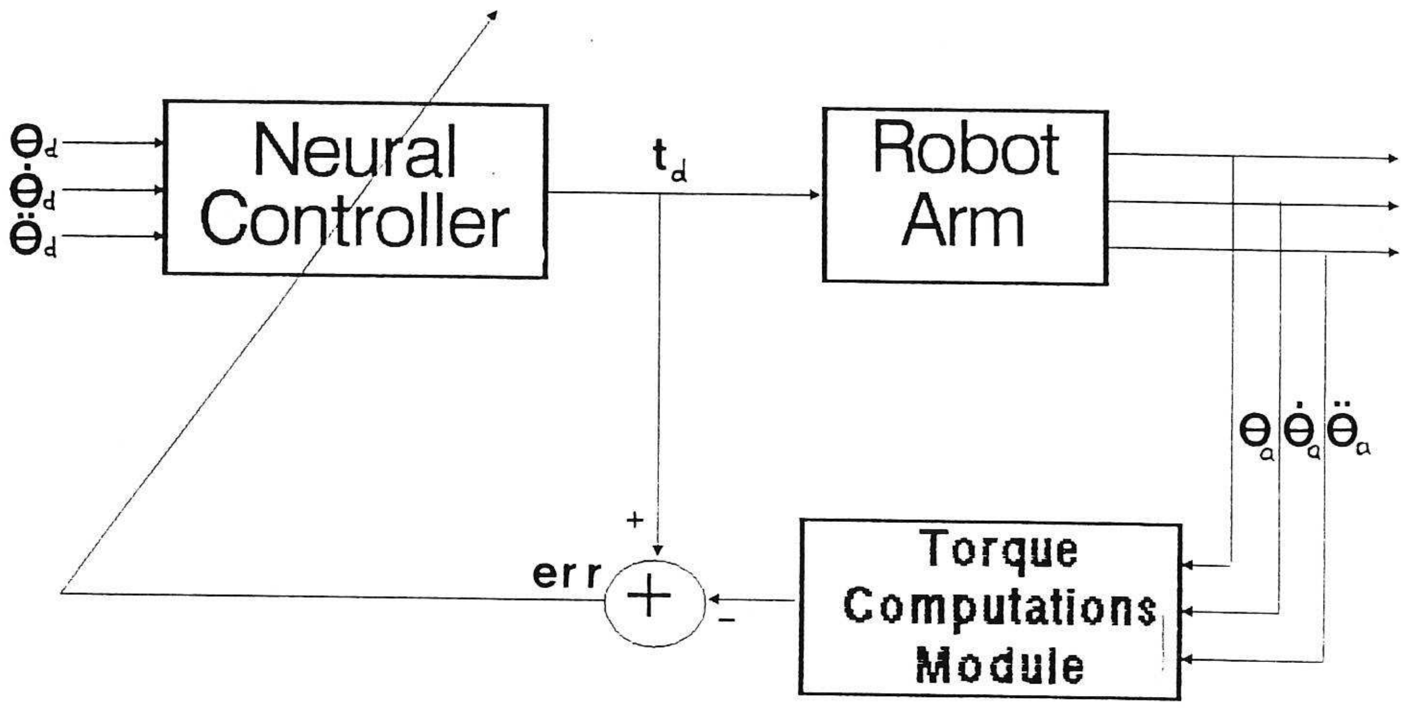

Figure (3) : The Robot Neural Controller

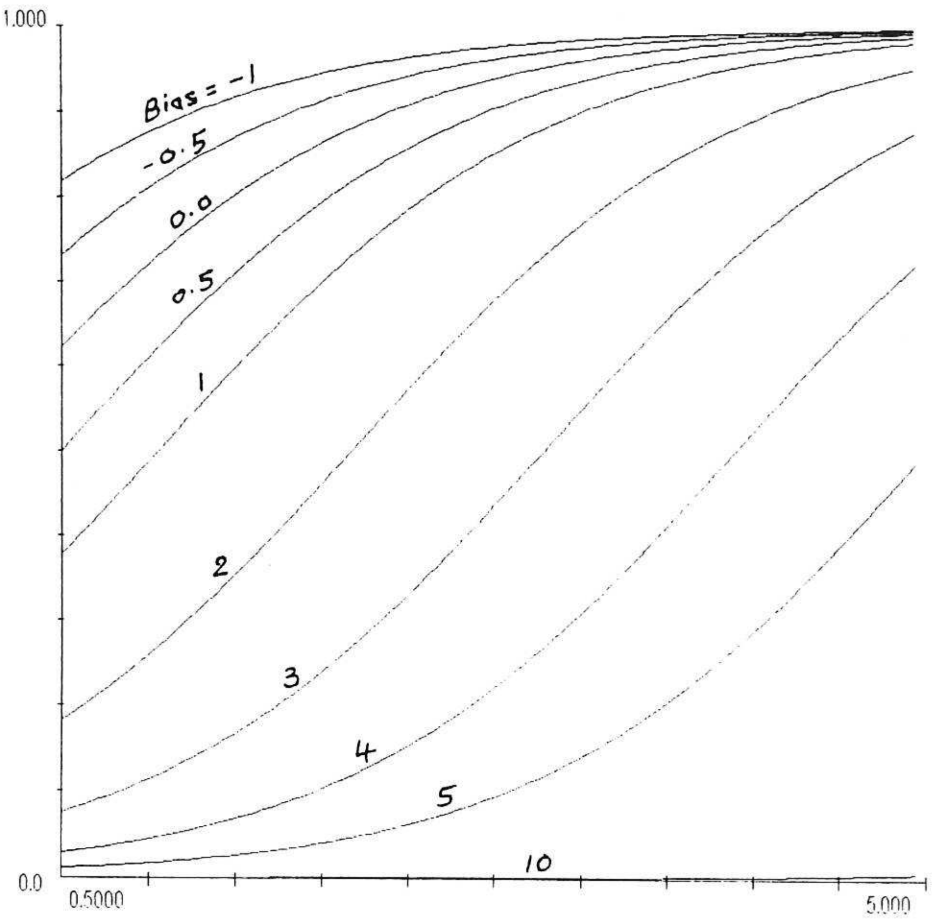

Figure (4) : The Sigmoid Function

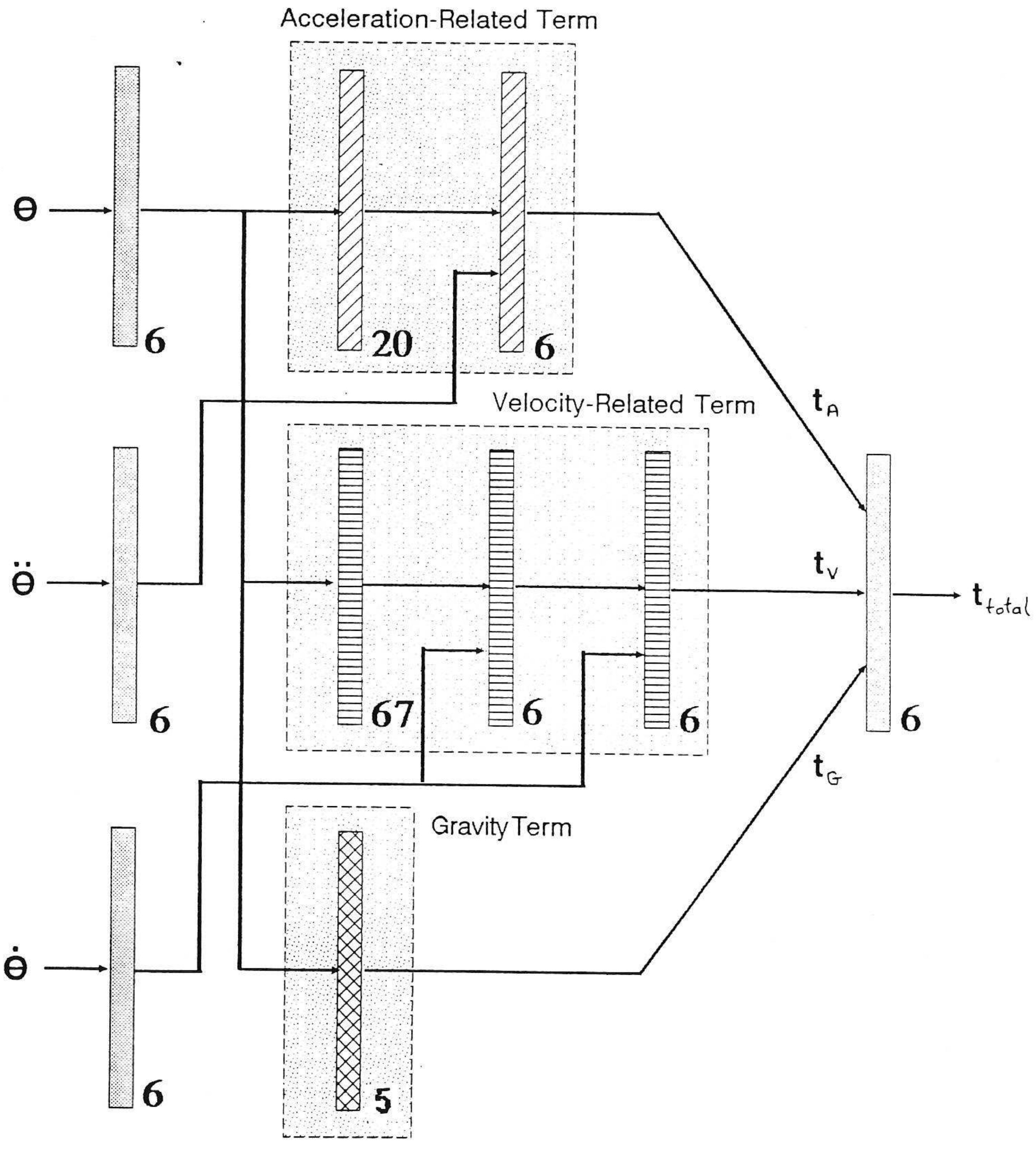

Figure (5) : The Complete Neural Network