

# Linear Hopfield Networks and Constrained Optimization<sup>1</sup>

G.G. Lendaris<sup>2</sup>, K. Mathia, and R. Saeks  
Accurate Automation Corp.  
7001 Shallowford Rd.  
Chattanooga, TN 37421

## Abstract

It is shown that a Hopfield neural network (with linear transfer functions) augmented by an additional feedforward layer can be used to compute the Moore-Penrose Generalized Inverse of a matrix. The resultant augmented linear Hopfield network can be used to solve an arbitrary set of linear equations or, alternatively, to solve a constrained least squares optimization problem. Applications in signal processing and robotics are considered. In the former case the augmented linear Hopfield network is used to estimate the "structured noise" component of a signal and adjust the parameters of an appropriate filter on-line, and in the latter case it is used to implement an on-line solution to the inverse kinematics problem via a Jacobi algorithm.

## 1 Introduction

The transfer functions of the computing elements in the class of neural networks called Hopfield Nets have traditionally been non-linear (step and sigmoid types) [8][9][10][11]. More recently, the non-linear transfer function has been replaced by a linear one, and the resulting *Linear Hopfield Network* used as the basis for solving certain *least squares* optimization problems [12][16][18]. The purpose of the present paper is to generalize the class of least squares optimization problems to which the Hopfield network can be applied, and, in particular, to demonstrate the applicability of these networks in **constrained** least-squares optimization problems. The key result is the demonstration that a Linear Hopfield Network (LHN) *augmented* with a pre- or post-feedforward layer can be configured to solve the constituent equations for such problems. More generally, such a network can be configured to solve an arbitrary set of linear equations with either real or complex coefficients [6][16][17][23]. In the present context the LHN can be considered an 'equation solver' (in contrast to an autoassociator), capable of solving systems of linear equations of the form  $Ax = b$ . The LHN employs an iterative process whose convergence to the equilibrium vector  $x_e$  implicitly inverts the square, positive-definite (and therefore invertible) system matrix  $A$ . The *augmented* LHN demonstrated herein implements a capability for dealing with

- 
1. This research was supported by the Naval Command, Control, and Ocean Surveillance Center, NR&D Division, under contract N66001-90-C-7021 and the Office of Naval Research under contract N00014-91-C-0268.
  2. On Sabbatical Leave from Portland State University, Portland, OR, 1993-94.

*non-square* and *non-invertible*  $A$ . If  $A$  is known to have full column rank, the “left inverse” of  $A$  is implemented; if  $A$  is known to have full row rank, its “right inverse” is implemented. Moreover, even with no *a priori* information about the rank of  $A$ , the Moore-Penrose *generalized inverse* (a “best” least-squares optimization procedure) can be implemented via the LHN. Each of these is accomplished by adding a feedforward layer either before or after the LHN.

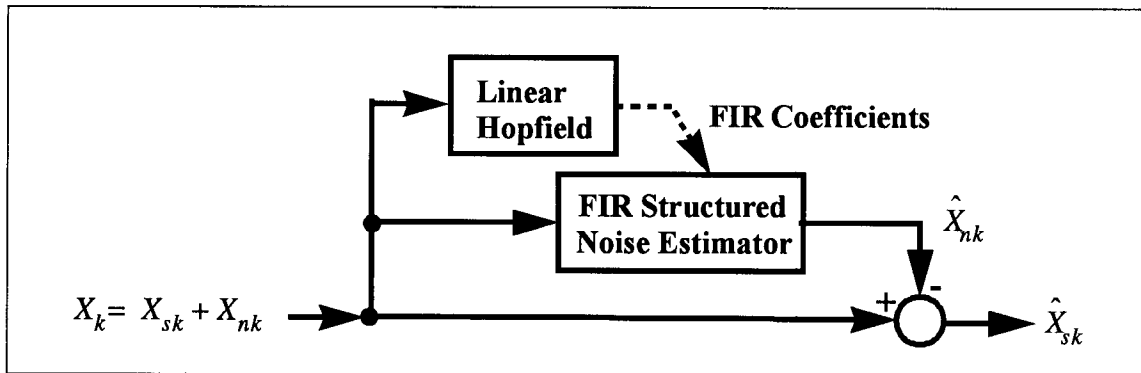
Constrained optimization problems abound in the world of applications, and two areas to which the above has been applied are described, namely, a filtering problem and an inverse kinematics problem. These are discussed in Sections 3 and 4 respectively. The problem context for each is developed, and in particular, the *constrained optimization* aspects of the problems are defined, and the procedure for using LHNs to solve them are given. We provide an overview of the two application examples in the remainder of the present section.

### 1.1 Optimal Filtering

For the **optimal filtering application**, refer to Figure 1. The input to the filter at time  $k$ ,  $X_k$ , comprises an information-bearing component (the “signal”),  $X_{sk}$ , and a structured noise component,  $X_{nk}$ . The latter is assumed to represent contaminating phenomena which can be modeled by a “low” order autoregressive formula (FIR model) [7].

$$\hat{X}_{nk} = \sum_{i=1}^m a_i X_{n(k-i)}$$

Depending on the application, the signal and autoregressive coefficients may be either real or complex.



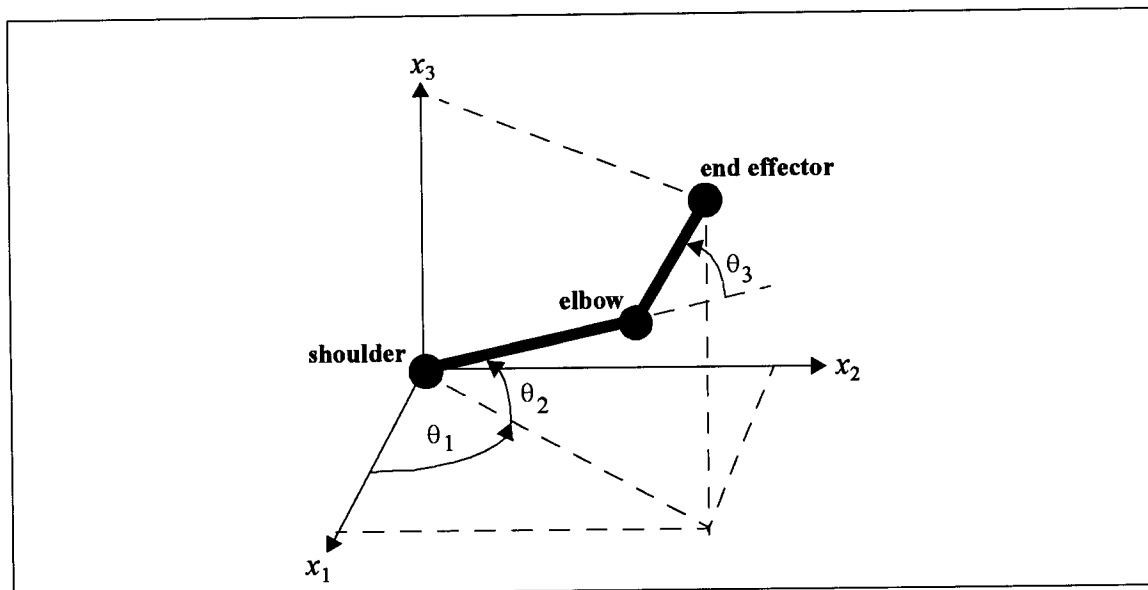
**Figure 1. Neural-network based adaptive filter to extract signal in context of structured noise.**

The problem is to perform an on-line estimate of the noise-process coefficients  $a_i$ , and feed them to a FIR filter which uses these coefficients to estimate the structured noise at time  $k$ . The resulting estimate  $\hat{X}_{nk}$  is then subtracted from the input  $X_k$  to (hopefully) yield a good estimate of the signal  $\hat{X}_{sk}$  at time  $k$ . An implementation using an (un-augmented) LHN to perform this task was reported in [18]. Unfortunately, while the approach yielded an excellent predictor of the structured-noise component, it often also contained a good prediction of the *signal* as well. So after the subtraction indicated in Figure 1 took place, there was not a significant improvement in

the signal-to-noise ratio. In the present paper we report an improved signal-to-noise ratio via using an augmented LHN. We formulate the problem to be solved with a *constraint* that the filter “ignore” certain known characteristics of the signal component [14]. This formulation leads to a non-positive-definite matrix to be inverted, hence the need for an augmented LHN. This hybrid neural network implements an algorithm which is guaranteed to yield the best (in the minimum-norm sense) least-squares solution for the set of autoregressive coefficients which optimally model the structured noise subject to the constraint: “ignore the signal”. This approach yields a filter with improved signal-to-noise ratio and real-time adaptation to changing noise conditions.

## 1.2 Inverse Kinematics

For the **inverse kinematics application**, refer to the diagram of Figure 2. Robot manipulators are usually required to track a prescribed end-effector trajectory. However, while the end-effector trajectory is typically given in Cartesian coordinates  $\mathbf{x} = [x_1, x_2, x_3]^T$ , the control system operates on the joint angles,  $\Theta = [\theta_1, \theta_2, \theta_3]^T$ , and therefore, a corresponding trajectory in  $\Theta$ -space must be computed. The two spaces are related by the coordinate transformation illustrated in Figure 2 via a set of nonlinear equations  $\mathbf{x}(t) = \mathbf{f}(\Theta(t))$ , where  $\mathbf{f}$  is defined by the forward kinematics of the robot manipulator [5].



**Figure 2.** Coordinate systems for a robot manipulator with 3 joint angles.

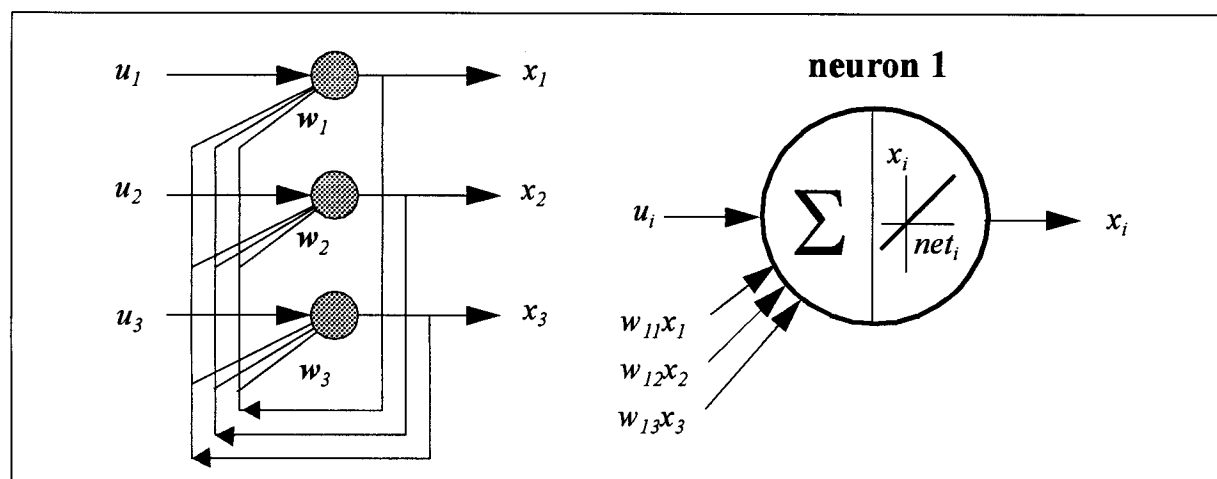
If the  $\Theta(t)$  trajectory were given, the forward computation of  $\mathbf{x}(t)$  would be a straightforward task; the inverse process of determining  $\Theta(t)$  from a given  $\mathbf{x}(t)$ , however, is substantially more difficult (known as the inverse kinematics problem). Moreover, when the robot has more than three degrees of freedom (joint angles), the equations are typically underdetermined and one must choose a trajectory in  $\Theta$ -space which is, in some sense, optimal among all possible solutions. A Jacobi technique is employed to set up a constrained linear optimization formulation of

the problem, and an augmented LHN is used to obtain the best (in the minimum-norm sense) least-squares solution for the set of joint angles for each step along the end-effector trajectory.

In the following section we review the required linear Hopfield concepts and formulate the *augmented* linear Hopfield networks required to solve the above described constrained optimization problems. Applications of this theory to the filtering and inverse kinematics problems are presented in Section 3 and Section 4, followed by our conclusions and an Appendix summarizing the various types of Hopfield networks and the role of the *linear* Hopfield network within that theory.

## 2 Linear Hopfield Networks

The Hopfield network, originally introduced in 1982 [8], is possibly the best known example of a recurrent neural network and is typically employed in one of two modes [15]: 1) to solve optimization problems (e.g., the well-known "traveling salesman" problem), or 2) as an autoassociator, more often in the latter mode. Although the concept was originally introduced as a continuous-time discrete-state system implemented by an analog electronic circuit, Hopfield and others extended the concept to include various combinations of continuous-state, discrete-state, continuous-time, and discrete-time systems [9][10][11]. More recently, it was observed that if one replaced the classical "squashing" transfer function with a linear transfer function, the resulting linear Hopfield network could be used to solve a least squares optimization problem -- all one need do is define a linear Hopfield network whose "energy function" is identical to the least squares performance measure one desires to minimize [6][12][16][17]. Unlike the classical Hopfield network with its "squashing" transfer function, in the linear case the values of the neuron states are not guaranteed to lie in a compact set and, therefore, an alternative stability theory is required. A summary of the various configurations of the Hopfield network appears in the Appendix, along with the corresponding stability and convergence conditions.



**Figure 3.** Topology of Hopfield networks, here with 3 neurons as an example. Linear transfer functions of its neurons characterize the linear Hopfield network.

The Linear Hopfield Network (LHN) is a single-layer, recurrent neural network whose neural elements have *linear* transfer functions (Figure 3). The outputs of the  $n$  elements comprise a state vector  $\mathbf{x}$  of dimension  $n$ , their connections are defined by a synaptic weight matrix  $\mathbf{W}$  of dimension  $(n \times n)$ , and the external input a vector  $\mathbf{u}$ , of dimension  $n$ . Each of  $\mathbf{x}$ ,  $\mathbf{W}$  and  $\mathbf{u}$  may be complex. For the purpose of the present paper we work with a continuous-state discrete-time linear Hopfield network defined by the (synchronous) update equation

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k + \mathbf{u}, \quad (1)$$

where the subscript  $k$  is an iteration designator. This network will be convergent if the spectral radius of its weight matrix,  $\rho(\mathbf{W})$ , is less than 1 (the eigenvalues of  $\mathbf{W}$  lie within the unit circle of the complex plane).

In this paper we are interested in applying LHNs to solving equations of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \quad (2)$$

Developing an LHN to solve such equations requires constructing the LHN's weight matrix  $\mathbf{W}$  in an appropriate manner to capture the information represented by  $\mathbf{A}$ . Wang and Li recently showed how Eq. (2) can be solved with full rank of  $\mathbf{A}$ , using a continuous-time, continuous-state LHN [23]. In the following we show how to extend this to an arbitrary  $\mathbf{A}$ , computing its generalized inverse and implementing the process with a discrete-time, continuous-state LHN.

## 2.1 Construction of LHN weight matrix $\mathbf{W}$ corresponding to system matrix $\mathbf{A}$

We start by rewriting Eq. (2) as follows:

$$-\mathbf{A}\mathbf{x} + \mathbf{b} = 0, \quad (3)$$

and with the motivation of using a LHN to solve it, we craft a (tentative) update rule of the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha(-\mathbf{A}\mathbf{x}_k + \mathbf{b}) \quad (4)$$

where  $\alpha$  is further crafted to ensure convergence of Eq. (4) (see below). In the limit as  $k$  increases (assuming convergence),  $\mathbf{x}_{k+1} = \mathbf{x}_k$ , so  $(-\mathbf{A}\mathbf{x}_k + \mathbf{b}) = 0$ , i.e., the solution being sought.

We rearrange Eq. (4) to

$$\mathbf{x}_{k+1} = (\mathbf{I} - \alpha\mathbf{A})\mathbf{x}_k + \alpha\mathbf{b}, \quad (5)$$

which has the form of the LHN definition in Eq. (1) with  $\mathbf{W} = (\mathbf{I} - \alpha\mathbf{A})$ . In order for the LHN to successfully solve Eq. (5), the term in brackets must satisfy the previously mentioned requirements on  $\mathbf{W}$ . This in turn *puts requirements on  $\mathbf{A}$*  (and on  $\alpha$ ) and if these requirements are not met, then suitable adjustments in problem formulation are required. We next consider a sequence of assumed attributes of  $\mathbf{A}$ , starting with the classical case of a symmetric positive definite  $\mathbf{A}$  and ending with an arbitrary rectangular  $\mathbf{A}$ , describing the corresponding formulation adjustments needed to solve Eq. (2) via an LHN.

**2.1.1 Assumption:  $A$  is square, hermitian and positive definite (invertible)**

A square, symmetric (hermitian in the complex case) and positive definite (invertible)  $A$  has positive real roots only, and thus clearly the spectral radius  $\rho(I-\alpha A)$  will be less than 1 if we choose  $\alpha$  such that

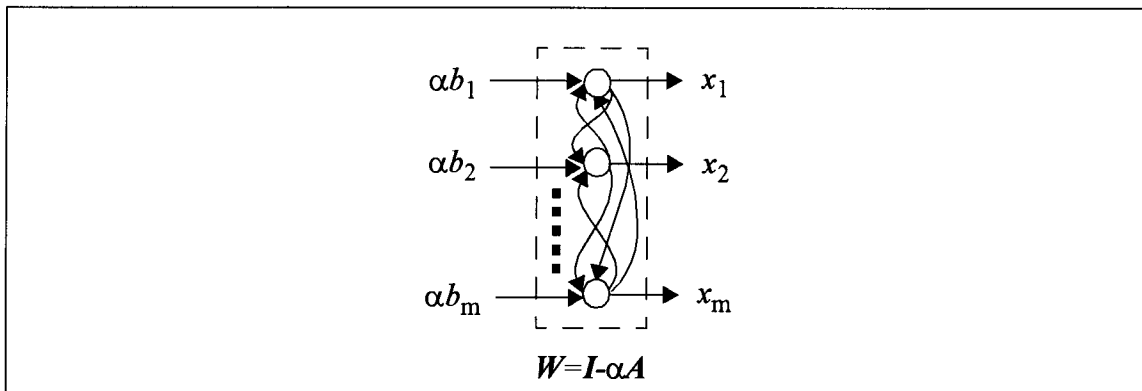
$$0 < \alpha < 2/\rho(A) . \tag{6}$$

With such a choice, the term in brackets in Eq. (5) will satisfy the requirements on  $W$  in Eq. (2). Since  $\rho(A) \leq \|A\|$  (for any norm), we could replace  $\rho(A)$  by  $\|A\|$  to yield the following bound for  $\alpha$ :  $0 < \alpha < 2/\|A\|$ . A particularly useful norm in this case is  $trace(A)$ , since it is easier to calculate than determining the eigenvalues of  $A$  needed for  $\rho(A)$ . We therefore use the bound  $0 < \alpha < 2/trace(A)$  in the remainder of the paper. Of course, a similar argument can be applied to the negative definite case.

Thus if  $A$  of Eq. (2) has the above properties and if  $\alpha$  is selected via Eq. (6), then an LHN with weight matrix  $W$  and input vector  $u$ ,

$$W = (I - \alpha A) , \quad u = \alpha b , \tag{7}$$

will converge to (the unique) solution of Eq. (2). There is motivation to select  $\alpha$  close to the upper limit, as this will speed convergence. A diagram for implementation of this network is shown in Figure 4.



**Figure 4.** Implementation of a linear Hopfield network to solve  $Ax=b$  for invertible, positive definite  $A$ . The weight matrix is  $W = I - \alpha A$ , the input vector is  $u = \alpha b$ .

**2.1.2 Assumption:  $A$  is square and invertible (but not necessarily hermitian or positive definite)**

Since  $A$  is not (necessarily) positive definite, convergence of Eq. (5) cannot be guaranteed by the above criteria. We invoke an algebraic fact that  $A$  is invertible if and only if  $A^*A$  is positive definite hermitian, and thereby set up a formalism where  $A^*A$  is to be inverted instead of  $A$  itself.

The equality  $A^{-1} = (A^*A)^{-1}A^*$  is used. (Throughout this paper,  $A^*$  represents the complex-conjugate transpose of  $A$ .)

Since  $A$  is invertible, we may write the solution of Eq. (2) directly as

$$\mathbf{x} = A^{-1}\mathbf{b} = (A^*A)^{-1}A^*\mathbf{b} . \quad (8)$$

Let  $B = A^*$  and  $\mathbf{d} = A^*\mathbf{b}$ , then Eq. (8) becomes .

$$\mathbf{x} = B^{-1}\mathbf{d}, \quad (9)$$

or, alternatively,  $-B\mathbf{x} + \mathbf{d} = \mathbf{0}$ , where  $B$  is hermitian, positive definite. Proceeding as we did from Eq. (3) to Eq. (5), we get an equivalent equation

$$\mathbf{x}_{k+1} = (I - \alpha B)\mathbf{x}_k + \alpha \mathbf{d}, \quad (10)$$

and a LHN with weight matrix  $W = (I - \alpha B)$  and  $0 < \alpha < 2/\text{trace}(B)$  will converge to the desired solution. Note that this is essentially the same process given in [23], except that a discrete-time implementation is used.

Eq. (10) provides a reference frame for reinterpreting the original problem of Eq. (2) for implementation via an LHN. The LHN has a weight matrix

$$W_{\text{hop}} = I - \alpha B = I - \alpha A^*A , \quad (11)$$

with  $0 < \alpha < 2/\text{trace}(A^*A)$ . Above we provided  $\alpha\mathbf{b}$  as the input vector to the LHN, now we must provide the vector  $\alpha\mathbf{d}$ , i.e., we must first perform the transformation  $\mathbf{d} = A^*\mathbf{b}$ . This is easily accomplished with a one-layer *feedforward* neural network whose inputs are the vector  $\mathbf{b}$ , and whose weight matrix is  $A^*$ . We also include the scale factor  $\alpha$  into this feedforward layer (ff) obtaining the weight matrix

$$W_{\text{ff}} = \alpha A^* . \quad (12)$$

A neural network that implements Eq. (10) comprises two stages:

*Stage 1:* A feedforward layer with input vector  $\mathbf{b}$  and weight matrix  $W_{\text{ff}} = \alpha A^*$ .

*Stage 2:* A linear Hopfield layer with weight matrix  $W_{\text{hop}} = I - \alpha A^*A$ , whose input vector is the output of the feedforward layer in Stage 1.

The structure of this augmented linear Hopfield network is shown in Figure 5a.

### 2.1.3 Assumption: $A$ is rectangular ( $n \times m$ ) with full column rank ( $m$ )

For this class of system matrices, we appeal to the theory of *generalized inverses* [19]. It is known that for rectangular  $A$  with full column rank, then  $A^*A$  of dimension ( $m \times m$ ) exists, its inverse exists, and there exists a matrix  $A_L^{-1} = (A^*A)^{-1}A^*$  which multiplies  $A$  on the left to

yield  $I_m$  (an identity matrix of dimension  $m$ ), and is called the left inverse of  $A$ . (This is equivalent to the equalities we invoked under Assumption 2.1.2 for a square matrix.)

Since a left inverse for  $A$  exists, we can write a solution of Eq. (2), if it exists, as:

$$x = A_L^{-1} b = (A^* A)^{-1} A^* b \quad (13)$$

As we did in Assumption 2.1.2 above, with  $B = A^* A$  and  $d = A^* b$  Eq. (13) becomes

$$x = B^{-1} d \quad (14)$$

or, alternatively,  $-Bx + d = 0$ , where  $B$  is hermitian and positive definite.

Proceeding as above (except in this case care is taken to keep track of the  $m$  and  $n$  dimensions) we obtain

$$x_{k+1} = (I_m - \alpha A^* A) x_k + \alpha A^* b, \quad (15)$$

where  $0 < \alpha < 2/\text{trace}(A^* A)$ . A neural network that implements Eq. (15) comprises two stages:

*Stage 1:* A feedforward layer with input vector  $b$  and weight matrix  $W_{ff} = \alpha A^*$ .

*Stage 2:* A linear Hopfield layer with weight matrix  $W_{hop} = I_m - \alpha A^* A$  whose inputs are the outputs of the feedforward layer in Stage 1.

The structure of this augmented linear Hopfield network is (also) shown in Figure 5a.

#### 2.1.4 Assumption: $A$ is rectangular ( $n \times m$ ) with full row rank ( $n$ )

We again appeal to the theory of *generalized inverses* [19], and note that for rectangular  $A$  with full row rank, then  $AA^*$  of dimension ( $n \times n$ ) exists, its inverse exists, and there exists a matrix  $A_R^{-1} = A^* (AA^*)^{-1}$  which multiplies  $A$  on the right to yield  $I_n$ , and is called the right inverse of  $A$  ( $I_n$  is the identity matrix of dimension  $n$ ).

Since the object  $A_R^{-1}$  exists, we can write

$$x = A_R^{-1} b = A^* (AA^*)^{-1} b, \quad (16)$$

Moreover, since  $x$  is in the range of  $A^*$  and  $A^*$  has full column rank (since  $A$  has full row rank)

there exists a unique  $x'$  such that  $x = A^* x'$ . Letting  $B = AA^*$ . Eq. (16) may then be expressed as

$$x' = B^{-1} b, \quad (17)$$

or, alternatively,  $-Bx' + b = 0$ , with  $B$  hermitian, positive definite. The counterpart of Eq. (15) for the present case is

$$x'_{k+1} = x'_k + \alpha (-Bx'_k + b) = (I_n - \alpha AA^*)x'_k + \alpha b \quad (18)$$

An LHN with weight matrix  $W_{\text{hop}} = I_n - \alpha AA^*$  and  $0 < \alpha < 2/\text{trace}(AA^*)$  will solve this equation, yielding the vector  $x'$ . Then, to obtain our desired vector  $x$ , we implement the transformation  $x = A^* x'$  with a feedforward layer whose weight matrix is  $\alpha A^*$  (this also captures the scaling factor  $\alpha$  which multiplies the input). A neural network that accomplishes this again comprises two stages:

*Stage 1:* A linear Hopfield layer with input  $b$  and with weight matrix  $W_{\text{hop}} = I_n - \alpha AA^*$ , yielding output  $x'$ .

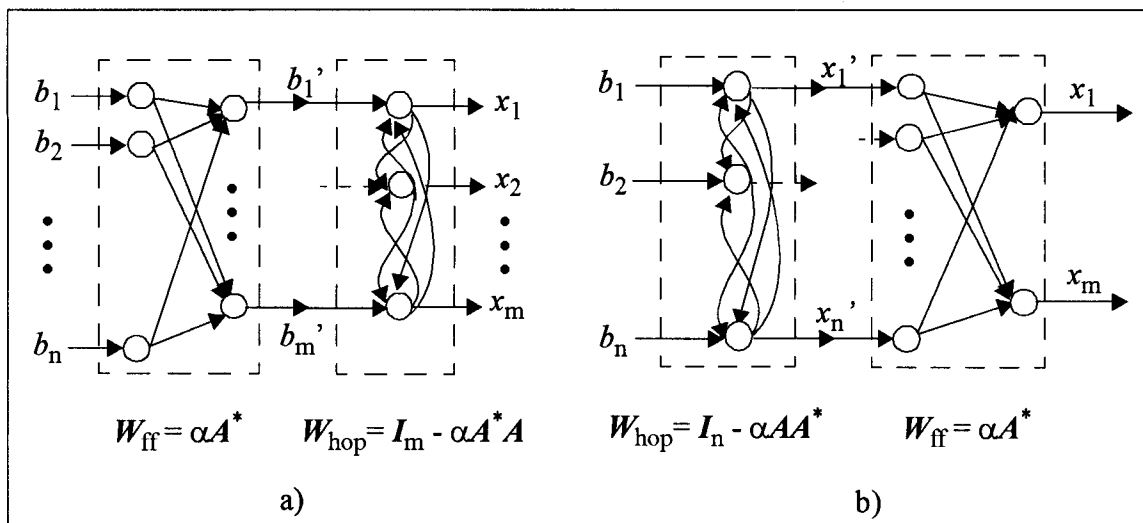
*Stage 2:* A feedforward layer with  $x'$  as its inputs and weight matrix  $W_{\text{ff}} = \alpha A^*$ , yielding output  $x$ .

The structure of this augmented linear Hopfield network is shown in Figure 5b.

Note that after Eq. (18), we could have taken a slightly different path. Namely, with  $x_k = A^* x'_k$ , we could multiply Eq. (18) by  $A^*$ , which with appropriate algebra, will yield

$$x_{k+1} = (I_m - \alpha A^* A) x_k + \alpha A^* b \quad (19)$$

This is identical to Eq. (15), and hence can also be implemented via the augmented LHN format shown in Figure 5a. Alternatively, in the development of Eq. (15), we could have made slightly different algebraic choices that would have led to Eq. (18). These considerations lead to the next, more general case which subsumes all previous cases.



**Figure 5. a) Linear Hopfield network (weight matrix  $W_{\text{hop}}$ ), augmented on left with a feedforward layer (weight matrix  $W_{\text{ff}}$ ). b) Linear Hopfield network, augmented on right with a feedforward layer.**

**2.1.5 Assumption:  $A$  is rectangular ( $n \times m$ ), no assumptions on rank (includes: rank  $< \min(n,m)$ )**

In the general case,  $A$  is a rectangular matrix of dimension ( $n \times m$ ) and there is no *a priori* information available about its rank. A *generalized inverse* of  $A$  is an ( $m \times n$ ) matrix  $G$  such that  $x = Gb$  is a solution of  $Ax = b$  for any  $b$  which makes the equation consistent. A variety of different generalized inverses have been defined [19]. We limit our discussion to the so-called Moore-Penrose inverse  $A^+$ . This generalized inverse is distinguished in that it provides the unique minimum-norm  $x$  which minimizes the error between  $Ax$  and  $b$  in a least square sense. The Moore-Penrose inverse (MP inverse) satisfies the following four properties [19]:

$$AA^+A = A, \quad A^+AA^+ = A^+, \quad AA^+ = (AA^+)^T, \quad \text{and} \quad A^+A = (A^+A)^T.$$

Under appropriate conditions (Assumption 2.1.1 to Assumption 2.1.4) the MP inverse reduces to the (special) inverses given above.

The following is extracted from a theorem and corollary in [19], pp. 62-63:

Let  $0 < \alpha < \frac{2}{c}$ , where  $c = \max(\text{eigenvalue of } A^*A)^2$ . Then the following equalities hold, with convergence assured:

$$A^+ = \alpha \sum_{k=0}^{\infty} (I_m - \alpha A^*A)^k A^*, \quad (20)$$

$$A^+ = \alpha \sum_{k=0}^{\infty} A^* (I_n - \alpha AA^*)^k, \quad (21)$$

where  $I_m$  is an identity matrix of dimension  $m$ , and  $I_n$  is an identity matrix of dimension  $n$ . The above provides a basis for constructing a LHN to compute the generalized inverse of  $A$ .

For arbitrary initial condition  $x_0$  Eq. (1) expands to:

$$\begin{aligned} x_1 &= Wx_0 + u \\ x_2 &= Wx_1 + u = W^2x_0 + Wu + u \\ x^k &= W^kx_0 + \sum_{j=0}^{k-1} W^j u \end{aligned}$$

In the limit, as  $k \rightarrow \infty$ , the LHN with fixed input  $u$  implements the matrix operation

$$\left( \sum_{j=0}^{\infty} W^j \right) u, \quad (22)$$

(which is independent of  $x_0$ , since the initial condition term drops out due to the convergence conditions on  $W$ ). Thus, for rectangular  $A$  with arbitrary rank, the best approximate solution of Eq. (2) in a least square sense can be formulated as

$$x = A^+ b = \alpha \sum_{k=0}^{\infty} A^*(I_n - \alpha A A^*)^k b = \alpha \sum_{k=0}^{\infty} (I_m - \alpha A^* A)^k A^* b, \quad (23)$$

We use Eq. (22) to construct hybrid neural networks for the two iterative solution processes defined by Eq. (23). The two implementations have the structures shown in Figure 5 and comprise a linear Hopfield layer and a feedforward layer.

A neural-network solution of the right-most expression in Eq. (23) will implement

$$x = M b', \quad (24)$$

where a *pre*-processing feedforward layer implements the transformation  $b' = \alpha A^* b$ , which is the input to a linear Hopfield layer that implements the iterative process

$$M = \sum_{k=0}^{\infty} W^k = \sum_{k=0}^{\infty} (I_m - \alpha A^* A)^k.$$

Of course, in an application the process would be stopped after a finite number of iterations, according to some convergence criterion. The implementation of Eq. (24) has a structure identical to the one developed in Figure 5a.

A neural-network solution of the middle expression in Eq. (23) will implement

$$x = \alpha A^* x', \quad (25)$$

where a linear Hopfield layer computes  $x' = N b$  by implementing the iterative process

$$N = \sum_{k=0}^{\infty} W^k = \sum_{k=0}^{\infty} (I_n - \alpha A A^*)^k.$$

The Hopfield component operates on the input  $b$ , yielding the  $x'$  as its output. Again, the iteration process would be stopped according to some convergence criterion. A *post*-processing feedforward layer computes the network output  $x$  using the transformation  $x = \alpha A^* x'$ . (Note that  $x'$  has dimension  $n$ , whereas  $x$  has dimension  $m$ ). The implementation of Eq. (25) has a structure identical to the one developed in Figure 5b.

## 2.2 General observations

The class of  $A$  matrices considered under Assumption 2.1.1 commonly arises in the solution of least squares optimization problems, such as that reported in [12]. This class of matrices requires the lowest computational load, as there are no additional matrix multiplications required (e.g., the  $A^*A$  calculation for the LHN weight matrix) nor are additional transformations needed (e.g., the feedforward layer). Thus there is incentive to formulate problems in such a way that this class of system matrices emerge. In this case a solution to Eq. (2) exists and it is unique, so there is no issue of choosing a “best” answer.

In cases where the system matrix  $A$  has properties discussed under Assumption 2.1.2 to Assumption 2.1.5, however, a unique solution may not exist. In this context, the Moore-Penrose (MP) *generalized inverse* represents a remarkable theory. The process involved in the MP generalized inverse is essentially one of double optimization: first, if an exact solution does not exist, then it finds those that are best approximate solutions in the least-squares sense, and second, it then selects the best solution in the minimum-norm sense from among the first set generated. The MP generalized inverse of a matrix always exists and is unique.

It can be shown that each of the inverses considered under Assumption 2.1.2 to Assumption 2.1.5 are specializations of the MP inverse [19]. Accordingly, since the MP inverse yields a “best-best” approximation to a solution in the general case, then so must the special inverses yield a “best” for their respective cases. As a metaphor, we might say that in the case of the right inverse, there is only one least-squares solution, so there is nothing for the second optimization to do. In the case of the left inverse, one particular solution is being hypothesized and tested, so there is nothing for the first optimization to do.

The MP inverse equations, Eq. (20) and Eq. (21), tell us that the MP inverse may be calculated in two distinct ways. Accordingly, this must be true also for each of the special cases. In the present context, the two computational methods are represented by the two augmented LHNs: one with a feedforward layer to the left of the LHN, and one with a feedforward layer to the right of the LHN. We showed the equivalence during our development of the right inverse in Assumption 2.1.4 above.

We note that while the two LHN configurations yield the same solution, distinctions between the two could be made based on computational considerations. For a matrix  $A$  of dimension  $(n \times m)$ , in the configuration using a pre-feedforward layer (Figure 5a), the dimension of the LHN layer is  $n$ , and in the configuration using a post-feedforward layer (Figure 5b), the dimension of the LHN is  $m$ . Since the most significant computations occur in the LHN, there is motivation to reduce its dimension, and thus select the right or left implementation according to whether  $n > m$  or  $m > n$ . (Since the LHN is capable of parallel implementation, if so implemented, time would not be a consideration, instead an allocation of resources issue.)

Finally, we note that though it is not likely, *if* it were possible to gain *a priori* knowledge of the rank of an  $(n \times m)$  matrix, where the  $\text{rank}(A) < \min(n,m)$ , then in principle it would be possible to set up a representation with a LHN whose dimension is the (assumed known) rank and augment this with *both* a pre-feedforward and post-feedforward layer to calculate a solution. As in

the previous paragraph, the motivation would be one of computational resources and/or time. Since such *a priori* knowledge is not likely in application, it won't be pursued further here.

An organization of the results in Section 2.1 is contained in Table 1 for easy reference. The entries are the recommended version of the augmented LHN structure determined by *a priori* information about the rank of  $A$ , whose dimension is  $(n \times m)$ . The criterion for each entry is a computational one, and is to assign  $\min(n,m)$  to the LHN (and hence  $\max(n,m)$  to the feedforward layer). We note that with no *a priori* information about rank of  $A$ , either augmented LHN may be used.

**Table 1: Appropriate Neural Network Configuration to Use According to Matrix Rank [ff=feedforward layer]**

	$A$ square	$A$ rectangular	$A$ rectangular	$A$ rectangular
	Positive Definite	Rank Full Row (n)	Rank Full Column (m)	Rank $< \min(n, m)$
ff is on <b>LEFT</b> of Hopfield			$\nabla$	$\nabla$ if $n \leq m$
ff is on <b>RIGHT</b> of Hopfield		$\nabla$		$\nabla$ if $n \geq m$
Hopfield only	$\nabla$			

### 3 Optimal Filtering Application

Our first example for application of the above ideas is a filtering problem, as indicated earlier in Figure 1. Recall that  $X_k$  represents the input to the filter at time  $k$ , comprising an information-bearing component (the "signal"),  $X_{sk}$ , and a structured noise component,  $X_{nk}$ . The latter is assumed to represent contaminating phenomena which can be modeled by an autoregressive formula (FIR filter) [7],

$$\hat{X}_{nk} = \sum_{i=1}^m a_i X_{n(k-i)}, \quad k \geq m + 1. \quad (26)$$

Depending on the application, the signal and autoregressive coefficients may be real or complex.

An adaptive filter of the above form has been developed for a tracking context [14] to distinguish targets from background and noise (the combination is here called structured noise). For this context the sequence  $X_k$  is complex, containing both in-phase and quadrature components,  $X_k = I_k + i \cdot Q_k$ , and the  $a_i$  of Eq. (26) are arbitrary complex coefficients. In the earlier work [18], the  $a_i$  coefficients were selected based only on information about the structured noise, with-

out taking into account any *a-priori* information that might be known about the signals representing the targets being tracked. Improvements to that design [7] take the next step of incorporating such *a priori* knowledge, via use of Lagrange Multiplier methods. This yields a constrained optimization formulation, and the augmented LHN described in the previous section is used to implement its solution.

Returning to a narrative of the problem, the task is to perform an on-line estimate of the noise-process coefficients  $\mathbf{a} = [a_1, a_2, \dots, a_m]^T$  that is optimal in some sense (see Eq. (27) below), and provide them to the FIR filter which uses these coefficients to estimate the structured noise at time  $k$ . The resulting estimate  $\hat{X}_{nk}$  is then subtracted from the input  $X_k$  to (hopefully) yield a good estimate of the signal  $\hat{X}_{sk}$  at time  $k$ . We formulate the problem to be solved with a *constraint* that the filter “ignore” certain known characteristics of the signal component. This formulation leads to a non-positive-definite matrix to be inverted, hence the need for an augmented LHN (as described in the previous section). This hybrid neural network implements a constrained least squares algorithm to obtain a set of autoregressive coefficients  $\mathbf{a}$  which continues to optimally model the structured noise, under the constraint: “ignore the signal”. This approach yields a filter with improved signal-to-noise ratio and real-time adaptation to changing noise conditions.

### 3.1 The Unconstrained Problem

It is desired to determine the coefficients  $\mathbf{a}$  for Eq. (26) that optimally model the data, based on minimizing

$$J(\mathbf{a}) = \frac{1}{2} \sum_{k=m+1}^M \|X_k - \hat{X}_k\|^2 = \frac{1}{2} \sum_{k=m+1}^M \left\| X_k - \sum_{i=1}^m a_i X_{k-i} \right\|^2 \quad (27)$$

over all possible complex coefficient vectors  $\mathbf{a} = [a_1, a_2, \dots, a_m]^T$ .

As usual, the minimization process proceeds by taking derivatives with respect to the  $a_p$ . We start by expanding Eq. (27):

$$J(\mathbf{a}) = \frac{1}{2} \sum_{k=m+1}^M (X_k - \hat{X}_k) (X_k - \hat{X}_k)^* = \frac{1}{2} \sum_{k=m+1}^M \left( X_k - \sum_{i=1}^m a_i X_{k-i} \right) \left( X_k^* - \sum_{j=1}^m a_j^* X_{k-j}^* \right)$$

and after (substantial) algebraic manipulation,

$$J(\mathbf{a}) = \frac{1}{2} \sum_{k=m+1}^M X_k X_k^* - \frac{1}{2} \sum_{j=1}^m a_j^* \sum_{k=m+1}^M (X_k X_{k-j}^*) - \frac{1}{2} \sum_{j=1}^m a_j \sum_{k=m+1}^M (X_k^* X_{k-j}) + \frac{1}{2} \sum_{i,j=1}^m a_i a_j^* \sum_{k=m+1}^M (X_{k-i} X_{k-j}^*)$$

Next, define

$$U_i = \sum_{k=m+1}^M (X_k X_{k-i}^*) \quad \text{and} \quad T_{ij} = \sum_{k=m+1}^M (X_{k-i} X_{k-j}^*) \quad , \quad (28)$$

yielding

$$J(\mathbf{a}) = \frac{1}{2} \sum_{k=m+1}^M X_k X_k^* - \sum_{j=1}^m \text{Re} [a_j U_j] - \frac{1}{2} \sum_{i,j=1}^m a_i a_j^* T_{ij} \quad (29)$$

We take derivatives of Eq. (29) with respect to the real and imaginary parts of the  $a_p$  separately. The first term of Eq. (29) drops out; the second term is easily evaluated to  $-\text{Re}[U_p]$ ; the third term is more tedious in that four cases of  $i,j$  vs.  $p$  must be considered. The result for the third term, making use of a substitution valid for the  $T$  matrix defined earlier:  $T_{ij} = \overline{T_{ji}}$ , is

$-\sum_{i=1}^m \text{Re} [T_{ip} a_i]$ . Thus we obtain

$$\frac{\partial J}{\partial \text{Re} [a_p]} = - \left( \text{Re} [U_p] + \sum_{i=1}^m \text{Re} [T_{ip} a_i] \right), \quad p = 1, 2, \dots, m \quad (30)$$

Similarly for the imaginary part,

$$\frac{\partial J}{\partial \text{Im} [a_p]} = - \left( -\text{Im} [U_p] + \sum_{i=1}^m \text{Im} [T_{ip} a_i] \right), \quad p = 1, 2, \dots, m \quad (31)$$

The negative sign is kept outside the bracket to indicate the negative direction of the gradient.

Using the now “classical” approach by Hopfield [8][9][10][11][22], one could proceed and define update equations from Eq. (30) and Eq. (31) for a Hopfield net consisting of  $2m$  nodes:  $m$  for the real terms, and another  $m$  for the imaginary terms,

$$\Delta \text{Re} [a_p] = \alpha \left[ \text{Re} [U_p] + \sum_{i=1}^m \text{Re} [T_{ip} a_i] \right], \quad p = 1, 2, \dots, m \quad (32)$$

and

$$\Delta \text{Im} [a_p] = \beta \left[ -\text{Im} [U_p] + \sum_{i=1}^m \text{Im} [T_{ip} a_i] \right], \quad p = 1, 2, \dots, m \quad (33)$$

where  $\alpha$  and  $\beta$  are appropriately “small” update coefficients.

In spite of the more complex derivation, the update formulae for the modified complex Hopfield network is essentially identical to the real case. The complex case differs in that only the real part of the excitation applied to  $a_p$  is used to update  $Re[a_p]$ , and only the imaginary part (with a sign change) is used to update  $Im[a_p]$ . If we view  $a_p = Re[a_p] + jIm[a_p]$  as a single complex update equation, and if we let  $\alpha = \beta$ , then we obtain

$$\Delta a_p = \alpha \left[ U_p^* + \sum_{i=1}^m T_{ip} a_i \right], \quad p = 1, 2, \dots, m \quad (34)$$

which has the same form as the classical Hopfield update formula [9][10], with the addition of the complex conjugate in Eq. (34) and in the definitions of  $U_p$  and  $T_{ip}$  (Eq. (28)).

Since (by the way it was defined) the matrix  $T$  is hermitian (i.e., equal to its complex conjugate transpose)  $J$  is a well defined "energy" function on its complex domain which is bounded below. As such, since  $\Delta Re[a_p]$  and  $\Delta Im[a_p]$  are defined to always reduce the value of  $J$ , the iterative process defined for our complex Hopfield network is guaranteed to converge.

Instead of following the "classical" approach after Eq. (31), we could proceed via the method suggested in the previous section in which a Linear Hopfield Network (LHN) is used to solve the (unconstrained, so far) equations of this subsection. Analytically, the next step would be to set Eq. (30) and Eq. (31) to 0. Equivalently, we could think in terms of setting the term in brackets in Eq. (34) to 0, i.e.,

$$U_p^* + \sum_{i=1}^m T_{ip} a_i = 0, \quad p = 1, 2, \dots, m \quad (35)$$

or, using vector notation,

$$U^* + Ta = 0 \quad (36)$$

where  $T$  is square, ( $m \times m$ ), hermitian, and positive definite. We are only a minus sign away from an equation of the form given in Eq. (3) and, as such, the linear Hopfield network described in Section 2.1.1 can be used to compute the AR coefficients required for our filter.

### 3.2 The Constrained Problem

While the above approach yielded an excellent predictor for the structured noise when applied in the tracking context [18], it often also contained a good prediction of the *signal* as well. Thus after the subtraction indicated in Figure 1 took place, there was not significant improvement in the signal-to-noise ratio.

In effect, by choosing the  $\mathbf{a}$  vector in the minimization process without information about the expected signal, a filter is obtained which optimally predicts the structured noise but is oblivious to the signal. Indeed, without use of *a priori* information about the signal, the manner in which the resulting predictor deals with the signal is not at all under control; it may ignore it or predict it perfectly.

To address this problem, a **constraint is added** to the process which, in effect, says “predict the structured noise as well as possible, being mindful of what is known about the expected signal(s).” To this end, it is here assumed that the signal from a target has *constant magnitude*, and a *phase that has a constant rate of change* over the extent of the target. This (assumed) *a priori* information is captured in the model

$$\hat{X}_{sk} = ce^{a+ibk}.$$

Although not precisely true, the constant (“dc”) value of the magnitude ( $c$  in the above equation) and the linear (affine) component of the phase typically dominate the return from a target. Moreover, as a first approximation, it is reasonable to assume that the phase is constant over the relatively short  $m$ -sample interval seen by the filter (i.e.,  $b$  is “small enough”), in which case the signal representing the target may be approximated by

$$\hat{X}_{sk} = ce^a = C.$$

Accordingly, a pure signal from a target would yield:

$$\sum_{i=1}^m a_i X_{s(k-i)} = \sum_{i=1}^m a_i C = C \sum_{i=1}^m a_i.$$

This is the *a priori* information to be used in the sequel. Based on this information, we select the  $\mathbf{a}$  vector to have the additional property

$$\sum_{i=1}^m a_i = 0. \tag{37}$$

Indeed, with this constraint on the  $a_i$ , the structured-noise predictor will ignore the signal part of the incoming sequence. Thus, we reformulate the problem as follows: **minimize Eq. (27) subject to the constraint that Eq. (37) holds.**

With the aid of the Lagrange Multiplier theorem, this constrained optimization problem can be converted into an equivalent unconstrained optimization that minimizes

$$J(\mathbf{a}) + \lambda \sum_{j=1}^m a_j \tag{38}$$

over all  $\mathbf{a}$  and  $\lambda$ .

Before proceeding with the development, we note that a side effect of these constraints is that the resulting filter would ignore the dc component of the structured noise as well as that of the signal. To accommodate this potential difficulty, the dc component of the structured noise is sub-

tracted off before computing coefficients for the filter and is then added back into the output of the prediction model. To accomplish this, the data used for developing the predictor coefficients is generated via:

$$\tilde{X}_k = X_k - X_{ndc}, \quad \text{where} \quad X_{ndc} = \frac{1}{M} \sum_{k=1}^M X_{nk}$$

and  $X_{nk}$  is a sequence containing structured-noise data only. In effect, the dc component of the structured noise is included in the predictor via analytical means, thereby eliminating the conflict between minimizing  $J(\mathbf{a})$  and satisfying the “no dc” constraint.

Continuing with the development of the constrained optimization via Eq. (38), we are again obliged to take derivatives. To facilitate this, let us rewrite Eq. (38) as follows:

$$L(\mathbf{a}) = J(\mathbf{a}) + \lambda_R \sum_{i=1}^m \text{Re}[a_i] + \lambda_I \sum_{i=1}^m \text{Im}[a_i] \quad (39)$$

where the  $R$  and  $I$  subscripts indicate real and imaginary, respectively.

Taking derivatives with respect to the  $a_p$ , and using Eq. (30) and Eq. (31), we obtain

$$\frac{\partial L}{\partial \text{Re}[a_p]} = \frac{\partial J}{\partial \text{Re}[a_p]} + \lambda_R = - \left( \text{Re}[U_p] + \sum_{i=1}^m \text{Re}[a_i T_{ip}] - \lambda_R \right) \quad (40)$$

$$\frac{\partial L}{\partial \text{Im}[a_p]} = \frac{\partial J}{\partial \text{Im}[a_p]} + \lambda_I = - \left( -\text{Im}[U_p] + \sum_{i=1}^m \text{Im}[a_i T_{ip}] - \lambda_I \right) \quad (41)$$

$$\frac{\partial L}{\partial \lambda_R} = - \left( - \sum_{i=1}^m \text{Re}[a_i] \right) \quad (42)$$

$$\frac{\partial L}{\partial \lambda_I} = - \left( - \sum_{i=1}^m \text{Im}[a_i] \right) \quad (43)$$

We set the above four equations to 0, and express them as follows:

$$\mathbf{Bq} + \Theta = 0, \quad (44)$$

where:

$$\mathbf{q} = \begin{bmatrix} \text{Re}[a_1] \\ \text{Re}[a_2] \\ \vdots \\ \text{Re}[a_m] \\ -\text{Im}[a_1] \\ -\text{Im}[a_2] \\ \vdots \\ -\text{Im}[a_m] \\ \lambda_R \\ \lambda_I \end{bmatrix}, \quad \mathbf{B}^* = \mathbf{B} = \begin{bmatrix} \text{Re}[T_{11}] & \dots & \text{Re}[T_{1m}] & \text{Im}[T_{11}] & \dots & \text{Im}[T_{1m}] & -1 & 0 \\ \text{Re}[T_{21}] & \dots & \text{Re}[T_{2m}] & \text{Im}[T_{21}] & \dots & \text{Im}[T_{2m}] & -1 & 0 \\ \vdots & & \vdots & & & & \vdots & \\ \text{Re}[T_{m1}] & \dots & \text{Re}[T_{mm}] & \text{Im}[T_{m1}] & \dots & \text{Im}[T_{mm}] & -1 & 0 \\ -\text{Im}[T_{11}] & \dots & -\text{Im}[T_{1m}] & \text{Re}[T_{11}] & \dots & \text{Re}[T_{1m}] & 0 & -1 \\ -\text{Im}[T_{21}] & \dots & -\text{Im}[T_{2m}] & \text{Re}[T_{21}] & \dots & \text{Re}[T_{2m}] & 0 & -1 \\ \vdots & & \vdots & & & & \vdots & \\ -\text{Im}[T_{m1}] & \dots & -\text{Im}[T_{mm}] & \text{Re}[T_{m1}] & \dots & \text{Re}[T_{mm}] & 0 & -1 \\ -1 & \dots & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & \dots & 0 & -1 & \dots & -1 & 0 & 0 \end{bmatrix}, \quad \Theta = \begin{bmatrix} \text{Re}[U_1] \\ \text{Re}[U_2] \\ \vdots \\ \text{Re}[U_m] \\ -\text{Im}[U_1] \\ -\text{Im}[U_2] \\ \vdots \\ -\text{Im}[U_m] \\ 0 \\ 0 \end{bmatrix}.$$

$T$  and  $U$  are defined in Eq. (28). Eq. (44) serves as the basis for implementing a LHN of  $2m+2$  neurons, with each neuron's output a real number, but representing the real and imaginary components as specified by the equation. We note, however, that  $\mathbf{B}$  is not positive definite, but it is hermitian [14]. We now have an example of Assumption 2.1.2 in the previous section, and an **augmented LHN** may be used to calculate a solution of Eq. (44), e.g., of the form shown in Figure 5a. Accordingly, the Figure 5a type implementation would have a feedforward layer with weight matrix  $\mathbf{W}_{\text{ff}} = \alpha \mathbf{B}^*$ , a linear Hopfield layer with  $\mathbf{W}_{\text{hop}} = (\mathbf{I}_m - \alpha \mathbf{B}^* \mathbf{B})$ , and auto-regressive coefficients given by the first  $2m$  components of the steady-state values of  $\mathbf{q}$ .

As an alternative representation, the real and imaginary components in the above vectors and matrix can be recomposed into complex numbers, yielding  $m+1$  dimensions instead of  $2m+2$ . Then we could construct an LHN with  $m+1$  *complex neurons*, each essentially operating separately on the real and imaginary parts of their incoming signals.

$$\text{We define } \mathbf{U}_e = \begin{bmatrix} \mathbf{U} \\ 0 \end{bmatrix}, \quad \mathbf{T}_e = \begin{bmatrix} \mathbf{T} & \mathbf{1} \\ \mathbf{1}^* & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{a}_e = \begin{bmatrix} \bar{\mathbf{a}} \\ -\bar{\lambda} \end{bmatrix}.$$

where the subscript 'e' denotes 'extended,'  $\mathbf{1}$  is an  $m$ -dimensional column vector of 1's, and  $\bar{\mathbf{a}}$  is the complex-conjugate of the vector  $\mathbf{a}$ . Each of  $\mathbf{U}$ ,  $\mathbf{T}$  and  $\mathbf{a}$  comprise complex numbers. The new equations are

$$\mathbf{T}_e \mathbf{a}_e + \mathbf{U}_e = 0. \tag{45}$$

As before we have a situation where the system matrix, in this case  $\mathbf{T}_e$ , is hermitian but not positive definite, and hence an **augmented LHN** is needed for calculating a solution of Eq. (45).

## 4 Inverse Kinematics Application

The second application is a tracking control system for a robot manipulator. For such manipulators, the control objective is usually to track a prescribed end-effector trajectory, where the trajectory is typically given in Cartesian coordinates  $\mathbf{x} = [x_1, x_2, x_3]^T$ . Unfortunately, the control system operates on the joint angles  $\Theta = [\theta_1, \theta_2, \theta_3]^T$ , and therefore, we are obliged to compute a corresponding trajectory in  $\Theta$ -space to provide to the controller. Figure 2 illustrates the relationships between the two coordinate systems for a manipulator with three adjustable joint angles.

Manipulator *kinematics* take into account the geometry of robot arm motions (as opposed to their dynamics). In mathematical terms, the manipulator's *forward* kinematics define the mapping from joint-angle space to Cartesian space, i.e. from control inputs to end-effector position. The trajectories in the two reference frames are related by

$$\mathbf{x}(t) = f(\Theta(t)), \quad (46)$$

where  $f(\cdot)$  is a set of nonlinear equations (defined by the forward kinematics),  $\mathbf{x}$  is typically a 3-dimensional vector and  $\Theta(t)$  is an  $n$ -dimensional vector, where  $n$  is the number of adjustable joint angles [5]. If the  $\Theta(t)$  trajectory were given, the forward computation of  $\mathbf{x}(t)$  would be a straightforward task; however, the *inverse* process of determining  $\Theta(t)$  from a given  $\mathbf{x}(t)$ ,

$$\Theta(t) = f^{-1}(\mathbf{x}(t)), \quad (47)$$

is substantially more difficult, both because of the numerical complexity of the inversion process and because the equations may be underdetermined. This is the problem we address.

A common method to facilitate the solution of a nonlinear system of equations along a trajectory is to linearize them about selected operating points, and solve the resulting linear equations for small excursions from the operating points [5]. Solutions of the inverse kinematics problem typically use this approach, where a linear mapping from velocities in joint angle space to velocities in Cartesian space is used as the basis for controlling robot manipulators. Thus the problem is treated via time derivatives, and we differentiate Eq. (46) to develop the velocity equation (or differential kinematics)

$$\frac{d\mathbf{x}}{dt} = \frac{d}{dt}[f(\Theta)] = \mathbf{J}(\Theta) \frac{d\Theta}{dt}, \quad (48)$$

where  $\mathbf{J}(\Theta)$  is the Jacobian matrix of  $f$  evaluated at  $\Theta$ . With  $\xi = \frac{d\mathbf{x}}{dt}$  and  $\omega = \frac{d\Theta}{dt}$  Eq. (48) takes the form of Eq. (2) and Eq. (3), respectively:

$$\mathbf{J}(\Theta) \omega = \xi. \quad (49)$$

The desire is to determine ("good") joint-angle velocities  $\omega$ , given the Cartesian velocities  $\xi$ .

## 4.1 The Unconstrained Problem

The joint space of a redundant robot manipulator has a higher dimensionality than the Cartesian target space. In this case the Jacobian  $J$  is not square, and one way to determine a "good"  $\omega = \dot{\Theta}$  via Eq. (49) is to set up a criterion function, and carry out an optimization process that will select the best  $\dot{\Theta}$  according to that criterion function [16]. Similar concepts were presented in [2][6]. Here we define the end-effector's position error at time  $t$ ,

$$\mathbf{x}_e(t) = \mathbf{x}(t) - \mathbf{x}_d(t), \quad (50)$$

where  $\mathbf{x}(t)$  is the actual end-effector position and  $\mathbf{x}_d(t)$  is the desired position. A typical criterion is to select  $\dot{\Theta}$  to minimize some measure of the position error. In addition, the designer may also desire to keep the joint-angle velocities small.

A criterion function that embodies these two desires is

$$E = \frac{1}{2} (\|\dot{\mathbf{x}}_e\|^2 + \varepsilon \|\dot{\Theta}\|^2), \quad (51)$$

where  $\varepsilon$  is an arbitrary non-zero, positive constant, which determines the 'importance' of the criterion "want small  $\dot{\Theta}$ ". Using Eq. (48), the criterion function becomes

$$\begin{aligned} 2E &= (J\dot{\Theta} - \dot{\mathbf{x}}_d)^T (J\dot{\Theta} - \dot{\mathbf{x}}_d) + \varepsilon \dot{\Theta}^T \dot{\Theta} \\ &= \dot{\Theta}^T A \dot{\Theta} - 2\mathbf{b}^T \dot{\Theta} + \dot{\mathbf{x}}_d^T \dot{\mathbf{x}}_d, \end{aligned}$$

where

$$\mathbf{A} = \varepsilon \mathbf{I} + J^T J, \quad \mathbf{b} = J^T \dot{\mathbf{x}}_d. \quad (52)$$

and  $\mathbf{I}$  is the ( $n \times n$ ) identity matrix.

The objective is to minimize the criterion function at a sequence of discrete points along the specified trajectory  $\mathbf{x}_d$  to yield the 'best'  $\dot{\Theta}$ . Thus the optimization is done at each point  $\mathbf{x}_d(t)$ ; accordingly, physical time  $t$  is 'fixed,' and the operations take place in 'iteration time'  $\tau$ . The derivative of  $E$  with respect to  $\tau$  is

$$\frac{dE(\dot{\Theta})}{d\tau} = \frac{1}{2} \frac{\partial E(\dot{\Theta})}{\partial \dot{\Theta}} \frac{d\dot{\Theta}}{d\tau} = (\mathbf{A}\dot{\Theta} - \mathbf{b}) \frac{d\dot{\Theta}}{d\tau}.$$

At the equilibrium this derivative becomes zero. Therefore

$$-\mathbf{A}\dot{\Theta} + \mathbf{b} = 0, \quad (53)$$

which is of the form of Eq. (3) and hence a LHN per Section 2 can apply. Since the scalar  $\varepsilon$  is greater than zero,  $\mathbf{A}$  is a square, symmetric and positive definite ( $n \times n$ ) matrix, and we refer to

Assumption 2.1.1 to design a LHN using Eq. (6) and Eq. (7). Note, the velocity term in Eq. (51) is necessary to guarantee a well defined optimization problem.

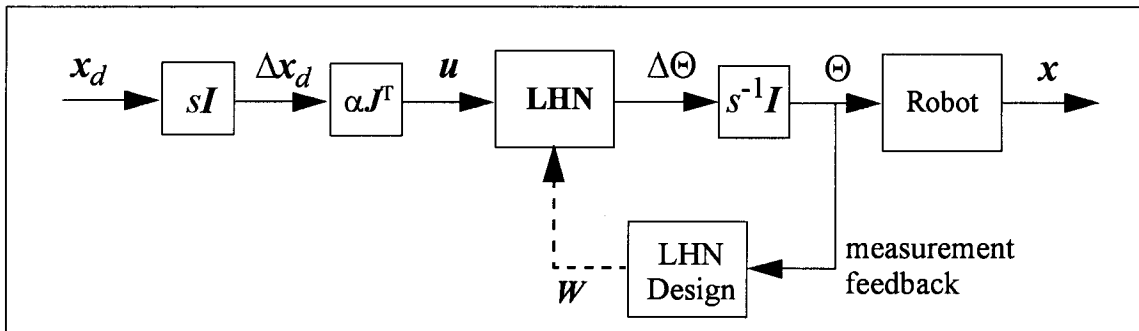
The resulting LHN has a structure as shown in Figure 4, here with weight matrix  $W$  and network input  $u$  given by

$$W = I - \alpha A = I - \alpha (\epsilon I + J^T J), \quad (54)$$

$$u = \alpha b = \alpha J^T \dot{x}_d. \quad (55)$$

In applications using digital computers the desired position trajectory  $x_d$  is given as a finite number of discrete-time points, therefore the desired velocity trajectory  $\dot{x}_d$  in Eq. (55) must be represented as the numerical time derivative  $\Delta x_d(t) = f_s [x_d(t+1) - x_d(t)]$ , where  $f_s$  is the sampling frequency. Using points of this time series as input vectors to the LHN, the final neuron states will represent the joint angle velocities  $\Delta \Theta(t)$ . The weight matrix of the linear Hopfield network must be re-specified at all points of the desired trajectory  $x_d$ , using Eq. (54) and Eq. (55).

Note that the transformation  $\alpha J^T \dot{x}_d$  could be implemented with an additional feedforward layer, giving a network structure similar to Figure 5a. A block diagram of the LHN-based inverse kinematics is shown in Figure 6.

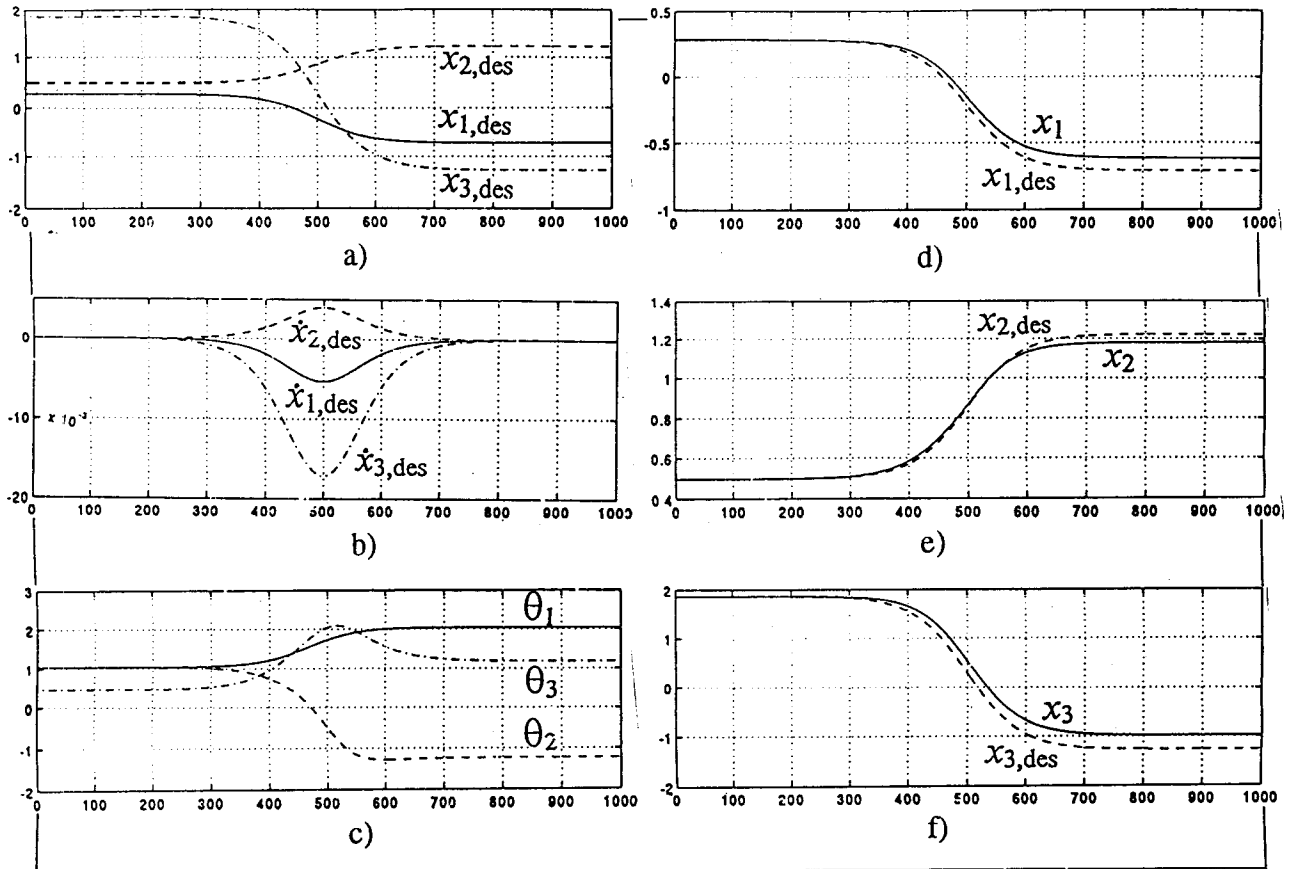


**Figure 6.** Block diagram of the inverse kinematics control using the criterion function approach and discrete-time signal vectors. The term  $sI$  represents a diagonal matrix with Laplace operators  $s$  as elements.

The objective of the optimization process is to minimize end-effector velocity and, to some degree, joint angle velocities. This formulation, while accomplishing the need of making the system matrix invertible, never the less, **compromises the velocity error**. Simulation results using this criterion function approach are shown in Figure 7. Note the offset of the actual end-effector trajectory from the desired trajectory.

## 4.2 The Constrained Problem

An alternative formulation of the inverse kinematics problem is: "Given the desired Cartesian trajectory  $x_d$ , find the minimum velocity joint angles  $\dot{\Theta}$  that minimize the end-effector's



**Figure 7. Simulation results for the criterion function approach. a) desired Cartesian end-effector trajectory, b) desired velocity profile for the end-effector, c) joint angle trajectory obtained by the LHN. The resulting actual end-effector trajectory in Cartesian space: d)  $x_1(t)$ , e)  $x_2(t)$ , f)  $x_3(t)$ . Note the steady position error, mainly caused by the criterion “small joint angle velocities”.**

tracking error  $x_e(t) = x(t) - x_d(t)$  (in a least mean square sense)!” This is a *constrained optimization* problem which can be solved by the iterative process defined by Eq. (23), obtaining the Moore-Penrose generalized inverse. The so-called ‘generalized-inverse control’ of robot manipulators has been analyzed intensively, see for example [1][13][21]. Here we apply an augmented linear Hopfield network to the generalized-inverse control of a robot manipulator, as also reported in [17].

Instead of providing an invertible optimization problem as in Section 4.1, we assume no *a priori* information about the Jacobian and use the full generalized inverse method as suggested in Assumption 2.1.5. We rewrite Eq. (49) as

$$-J(\Theta)\omega + \xi = 0, \quad (56)$$

which is of the form of Eq. (3). Accordingly, the solution to Eq. (50) can be obtained via an **augmented LHN** with either of the two structures given in Figure 5. We choose a network with the structure in Figure 5a to implement the process in Eq. (23). The network has weight matrices

$$W_{\text{ff}} = \alpha J^T, \quad W_{\text{hop}} = I - \alpha J^T J,$$

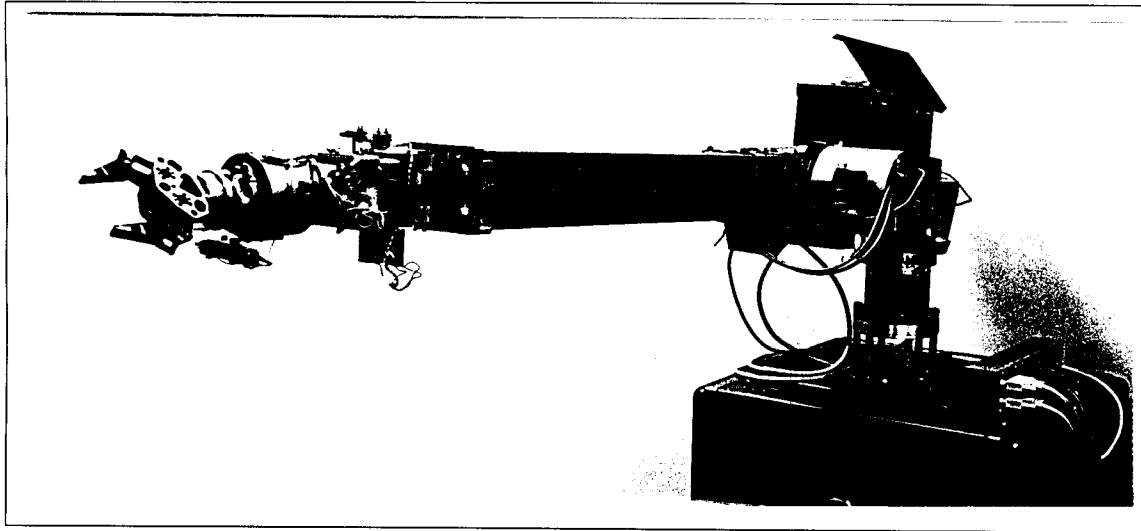
where  $J$  may be singular (since  $J$  is real, the hermitian conjugate operator  $*$  is simply a transpose operator). The input to the augmented LHN ( $\xi$  in Eq. (56)) is the desired discrete-time Cartesian velocity at time  $t$ ,  $\Delta x_d(t)$ . First the feedforward layer computes the input to the subsequent LHN,  $\Delta x'_d = W_{\text{ff}} \cdot \Delta x_d$ . The LHN then converges to the 'best' joint angle velocities  $\Delta \Theta = W_{\text{hop}}^{-1} \cdot \Delta x'_d$  that maintain the desired Cartesian velocity trajectory, thus implicitly inverting the weight matrix,  $W_{\text{hop}}^{-1}$ . Combined, the two-stage process computes the optimal joint angle velocities in a least square sense, as stated in Section 2.1. The integrated angle velocities give the angles required to control the robot manipulator (Figure 9). This procedure is repeated for each point along the desired Cartesian trajectory.

Recall that the states of Hopfield networks slide towards the bottom of an "energy" well. In our case, after the network has converged to the minimum, a new weight matrix is specified and the optimization process is repeated. With a new weight matrix the energy well has "moved," and the network may be thought of as "chasing" the minimum of an energy well that moves according to a desired end-effector path. This method applies in particular when the system matrices (here the Jacobians) are not known *a priori*, e.g. when the trajectory is designed *on-line* as in obstacle avoidance. For real time applications it also requires extremely fast neural-network processing. Thus a key requirement is a parallel neural network implementation in hardware. Such a Neural Network Processor is available to the authors [14][20].

### 4.3 Implementation of the Manipulator Neurocontrol System

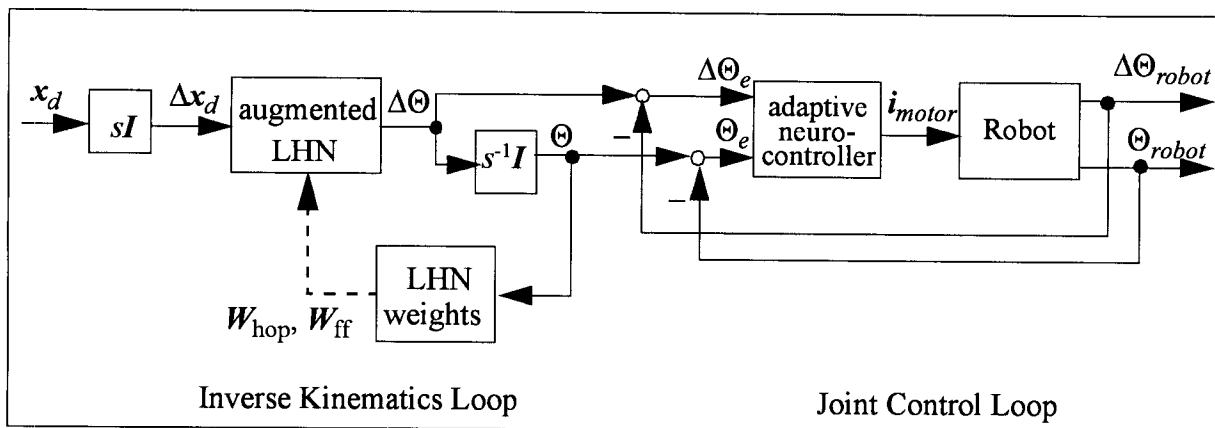
The LHN-based generalized inverse control was tested on an extendable stiff arm manipulator (ESAM), developed by NASA. The ESAM is shown in Figure 8 and has a structure similar to the one illustrated in Figure 2. Our neurocontrol system for robot manipulators includes two neural network based control loops (Figure 9): the first loop implements the LHN-based inverse kinematics process, the second loop includes an adaptive neural joint controller, which asymptotically minimizes remaining tracking errors to zero, as also reported in [3][4].

In the experiment reported here the desired position and velocity trajectories consisted of 4000 points each. Similar to the velocity profile in Figure 7b, it was desired that the end-effector start slowly, reach maximum speed at the trajectory midpoint and approach the final position with decreasing velocity. In order to evaluate the generalized inverse control, a 'desired' joint angle trajectory  $\Theta_d$  was constructed (Figure 10a) and mapped to the desired Cartesian trajectory  $x_d$  using the ESAM forward kinematics (Figure 10b). The  $\Theta$ -trajectory computed by the LHN inverse kinematics was compared with  $\Theta_d$  (Figure 10c), note the high accuracy. The end-effector's joint angle error  $\Theta_e$  is plotted in Figure 10d. Note that  $\Theta_e$  first increases because joint controller does not 'know' the manipulator characteristics, i.e. the controller is model free. After about six seconds the adaptation process is completed and the neurocontroller asymptotically reduces the tracking error to almost zero. The entire control system, including a graphical interface, neural



**Figure 8.** The 3-joint extendable stiff arm manipulator (ESAM), developed by NASA. The generalized-inverse control, based on an augmented linear Hopfield network, was verified on this robot.

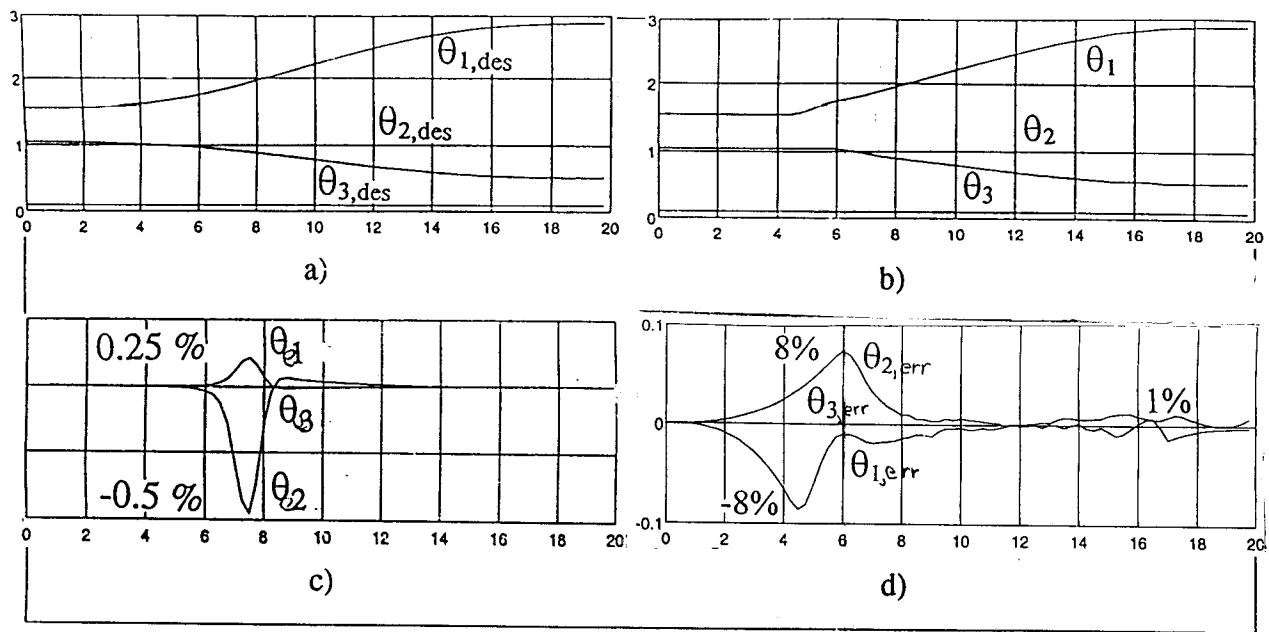
networks and control algorithms, was designed and implemented by Accurate Automation Corporation on a Silicon Graphics computer.



**Figure 9.** Block diagram of the neurocontrol system for tracking an end-effector trajectory. All signals are discrete-time vectors.

## 5 Conclusions

An enhanced version of the linear Hopfield network, capable of implementing a full Moore-Penrose Generalized Inverse for an arbitrary matrix, has been developed and applied to two on-line computational problems. In the case of the signal processing problem, structured noise is observed on-line and used to define the weights of an augmented linear Hopfield network which adaptively produces the coefficients of an appropriate filter. In the second example, an augmented linear Hopfield network is used to solve the inverse kinematics problem for a robot arm at each



**Figure 10. Applying the neurocontrol system in Figure 9 to the extendable stiff arm manipulator (joint angles in radians, time in seconds): a) 'desired' joint angle trajectory  $\Theta_d$  as constructed for testing, b) trajectory  $\Theta$  computed by the LHN-based inverse kinematics, c) inverse kinematics error  $\Theta_e = \Theta - \Theta_d$ , d) actual end-effector error  $\Theta_{err}$  performed by the ESAM. The initially increasing error is minimized by the adaptive neurocontroller.**

point along a specified end-effector trajectory. Note, in both examples the power of the linear Hopfield network lies with the fact that the data which defines the problem is received in real-time and the weights of the Hopfield network are adapted on-line. Indeed, if the required data was available a-priori one could compute the generalized inverse off-line and implement it with a simple feedforward network. For an on-line application, however, the weights of the linear Hopfield network can be readily updated at each time step and the Hopfield iteration can be efficiently implemented in parallel if necessary.

## 6 References

- [1] J. Ballieul, "Kinematic Programming Alternatives for Redundant Manipulators," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 722-728, 1985.
- [2] Y. Bestaoui, "An unconstrained optimization approach to the resolution of the inverse kinematic problem of redundant and nonredundant manipulators," *Robotics and Autonomous Systems*, Vol. 7, No. 1, pp. 37-45, March 1991.
- [3] C. Cox, J. Edwards, R. Saeks, R. Pap and K. Mathia, "Adaptive Semi-Autonomous Robotic Neurocontroller," *Proc. SPIE Conference - Applications of Artificial Neural Network V*, Orlando, Fl., April 1994.
- [4] C. Cox, K. Priddy, K. Mathia, R. Saeks and R. Pap, "Hardware Implementation of a Semi-

- Autonomous Robotic Neurocontroller," *Proc. U.S. Navy SBIR Conference - Neural Network Applications*, Arlington, Va., June 1994.
- [5] Craig, J.J., *Introduction to Robotics: Mechanics and Control*, Addison Wesley, 1989.
- [6] J. Guo and V. Cherkassky, "A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing," *Proc. Int. '1 Joint Conference on Neural Networks*, Washington, D.C., June 1989.
- [7] S. Haykin, W. Stehwien, C. Deng, P. Weber and R. Mann, "Classification of Radar Clutter in an Air Traffic Control Environment," *Proceedings of the IEEE*, Vol. 79, No. 6, pp. 742-772, 1991.
- [8] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. of the Nat. Acad. of Sci. USA*, Vol. 79, pp. 2554-2558, 1982.
- [9] J.J. Hopfield, "Neurons with Graded Response have Collective Computational Properties like those of Two-State Neurons", *Proc. of the Nat. Acad. of Sci. USA*, Vol. 81, pp. 3088-3092, 1984.
- [10] J.J. Hopfield and D.W. Tank, "Neural Computation of Decision Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.
- [11] J.J. Hopfield and D.W. Tank, "Computing with Neural Circuits: A Model," *Science*, vol. 233, pp. 625-633, August 1986.
- [12] Kelly, M.F., P.A. Parker and R.N. Scott, "Myoelectric Signal Analysis using Neural Networks", *IEEE Engineering in Medicine and Biology Magazine*, pp. 61-64, March 1990.
- [13] Klein, C.A., Huang C.H., "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," *IEEE Transaction on Systems, Man, and Cybernetics*, Vol. SMC-13, No. 3, pp. 245-250, 1983.
- [14] G.G. Lendaris, R.M. Pap, R.E. Saeks, C.R. Thomas and R.M. Akita, "Hardware Neural Network Implementation of Tracking System," *Proc. IEEE Workshop on Neural Networks for Signal Processing-1994*, Greece, September 1994.
- [15] A.J. Maren, C.T. Harsten and R.M. Pap, *Handbook of Neural Computing Applications*, Academic Press, 1990.
- [16] K. Mathia, R.E. Saeks, "Inverse Kinematics via Linear Dynamic Networks", *Proc. of World Congress on Neural Networks*, San Diego, June 1994.
- [17] K. Mathia, R.E. Saeks and G.G. Lendaris, "Linear Hopfield Networks, Inverse Kinematics and Constrained Optimization", *Proc. of IEEE Int'l Congress on Systems, Man and Cybernetics*, October 1994.
- [18] R.M. Pap, R.E. Saeks, C.R. Thomas and R.M. Akita, "Neural Network Implementation of Stochastic Filters for Radar Tracking," *Proc. of IEEE Workshop on Neural Networks for Signal Processing*, Helsingoer, Denmark, October 1992.
- [19] C.R. Rao and S.K. Mitra, *Generalized Inverse of Matrices and its Applications*, John Wiley & Sons, 1971.
- [20] R.E. Saeks, K. Priddy, R.M. Pap and S. Stowell, "On the Design of the MIMD Neural Network Processor," *Proc. of SPIE Symposium on Neural Networks V*, Orlando, June 1994.

- [21] L. Sciavicco and B. Siciliano, "A Solution Algorithm to the Inverse Kinematic Problem for Redundant Manipulators," *IEEE Journal of Robotics and Automation*, Vol. 4, No. 4, pp. 403-410, 1988.
- [22] D.W. Tank and J.J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", *IEEE Trans. on Circuits and Systems*, Vol. CAS-33, pp. 533-541, 1986.
- [23] J. Wang and H. Li, "Solving Simultaneous Linear Equations Using Recurrent Neural Networks," *Information Sciences*, Vol. 76, No. 3 and 4, pp. 255-277, 1994.

Appendix: Overview of Common Hopfield Network architectures

	Discrete States	Continuous States, Nonlinear	Continuous States, Linear
Discrete Time	<p>Hopfield Network defined by:</p> $net(n) = Wx(n) + u$ $x_i(n+1) = \begin{cases} 0 & \text{if } net_i(n) < 0 \\ x_i(n) & \text{if } net_i(n) = 0 \\ 1 & \text{if } net_i(n) > 0 \end{cases}$ <p>Synchronous or asynchronous dynamics (update) possible.</p> <p>Convergence if <math>W = W^T</math>.</p>	<p>Hopfield Network defined by:</p> $net(n) = Wx(n) + u$ $x(n+1) = g[net(n)]$ <p>Transfer function <math>g(\cdot)</math> is:</p> <ul style="list-style-type: none"> <li>• strictly increasing,</li> <li>• continuously differentiable,</li> <li>• bounded above and below.</li> </ul> <p>Convergence if <math>W = W^T</math>.</p>	<p>Hopfield Network defined by:</p> $net(n) = Wx(n) + u$ $x(n+1) = net(n)$ $W = I - \alpha A$ <p>Transfer function is linear.</p> <p>Convergence to desired solution if</p> $A = A^T > 0 \text{ and } 0 < \alpha < \frac{2}{\lambda_{max}}$
Continuous Time	<p>Hopfield Network defined by:</p> $net(t) = Wx(t) + u$ $x_i(t) = \begin{cases} 0 & \text{if } net_i(t) < 0 \\ x_i(t) & \text{if } net_i(t) = 0 \\ 1 & \text{if } net_i(t) > 0 \end{cases}$ <p>Network well defined, convergent if:</p> <ul style="list-style-type: none"> <li>• asynchronous dynamics (update),</li> <li>• <math>W = W^T</math>,</li> <li>• <math>w_{ii} = 0</math>.</li> </ul>	<p>Hopfield Network defined by:</p> $\frac{d}{dt}net(t) = -net(t) + Wx(t) + u$ $x(t) = g[net(t)]$ <p>Transfer function <math>g(\cdot)</math> is:</p> <ul style="list-style-type: none"> <li>• strictly increasing,</li> <li>• continuously differentiable,</li> <li>• bounded above and below.</li> </ul> <p>Network is (asymptotically) stable if and only if <math>W = W^T</math>.</p>	<p>Hopfield Network defined by:</p> $\frac{d}{dt}net(t) = -net(t) + Wx(t) + u$ $x(t) = net(t)$ $W = I - A$ <p>Transfer function is linear.</p> <p>Network is (asymptotically) stable if</p> $A = A^T > 0.$