Simulation of an overhead crane using neural networks

Goosen, P.C.

Award date:
1993

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
Measurement and Control Section

SIMULATION OF AN OVERHEAD CRANE USING NEURAL NETWORKS

by P.C. Goosen

Master of Science Thesis
project carried out from December 1992 to August 1993
commissioned by prof.dr.ir. P. Eykhoff
under supervision of dr. ir. A.A.H. Damen and ir. J. Mazák
date: August 1993

Goosen, P.C. Simulation of an Overhead Crane Using Neural Networks

M.Sc. Thesis, Measurement and Control Section (ER), Electrical Engineering, Eindhoven University of Technology, The Netherlands, August 1993.

ABSTRACT

The overhead crane is a nonlinear MIMO process. The control has to ensure that the load follows some specified trajectory. One of the main problems is the swinging of the load that has to be suppressed. In the experiments a computer simulation model of the crane is used.

We wish to design a neural network controller that uses a nonlinear neural network as a model of the crane. In this thesis the simulation experiments of the model neural network are described. The neural network is trained in a black box configuration using a random input- output data sequence, by minimizing the error between the system output and the neural network output.

The first results after training the network show that the swinging of the load is not represented by the neural network. During the training of the network different algorithms are tried to put more weight on the frequency corresponding to the swinging.

For the dynamics of the neural network besides the ARMA representation the Delta representation neural network model was tried.

It is shown that all the necessary information is contained in the dataset. We are convinced that the neural network is able to map the input- output relations better, though a full representation of the nonlinear behaviour requires a far more extensive neural net as is analysed. We conclude that the minimization algorithms, stochastic and gradient methods, are not powerful enough to find a proper neural network.

CONTENTS

CHAPTER 1
Introduction

In this Thesis we try to simulate an overhead crane with a neural network model. We want to use this model in a neural network controller.

An overhead crane is an equipment for transport of heavy loads over short distances. A trolley that moves along a rail hoists the cargo at one place and puts it on the end position. In a harbour it's usually applied to hoist containers from ships to trucks on the quay. The control problem is that the load has to follow a specified trajectory. Because of efficiency the transport has to be done in minimum time while at the end position the swinging of the load will have to be suppressed to within certain bounds, to be able to place the load accurately at the right place. Nowadays an overhead crane is usually controlled by an operator who needs a lot of experience to handle it properly.

A pilot scale model of an overhead crane will soon be available to the "System Identification for Control" (SIC) group of the measurement and control section. Until then we're still working with a computer simulation model of the crane. For the implementation of this model we need the physical equations describing the system.

Most of the methods using models for control require linear equations and can be applied only if the system to be controlled is linearized around some working point. Meanwhile actually most of the complex dynamic industrial processes show non linearities. So does the overhead crane. In recent years study has been done in the SIC group for non linear models for control and particularly for control using neural networks. A few identification experiments were done on relatively simple examples from the literature using neural networks besides linear models (Pijnenburg[13]). The identification of the overhead crane without hoisting was tried using the static back propagation algorithm (vdBeemt[1]). A SISO acoustic process was identified using dynamic back propagation (Moonen[7]). After that the software for dynamic back propagation was adapted to MIMO systems.

Algorithms that have been drafted and the conclusions that have been drawn from the experiments so far will be applied for simulation of the MIMO overhead crane. The system will be prestabilized with a linear controller leaving enough freedom for excitation for the overall neural network controller.

We will start with simulation experiments to find a proper model. After this a controller will have to be found.

CHAPTER 2
Description

The physical system of the overhead crane or gantry crane is also described by Schreppers[15] who applied a fuzzy logic controller to the same system. The formulas are repeated here briefly, then the special configuration in which the system will be used in our experiments is explained. But first attention is paid to controller configurations.

2.1 Controller Description

There are different configurations for control using a neural network as a model and also a neural network as a controller. Also structures are known in which the linear part of the system is represented by a linear model, see Narendra[10] and Youji[18]. This is used parallel to the nonlinear neural network model. So the neural network only represents the nonlinear terms. We are convinced that the neural network should be able to map both the linear and the nonlinear behaviour as will be explained in chapter three. From experiments we saw that the two descriptions were working against each other in the training phase, see vdBeemt[1]. An overview of some well known controller structures is given by Hunt [3]. Two possibilities are explained here.

2.1.1 Internal Model Control

Internal model control (IMC) is a well known scheme from linear control theory. It is explained by Morari[9]. Hunt[4] applies it for controllers using neural networks. In figure 2.1 the IMC configuration is plotted.

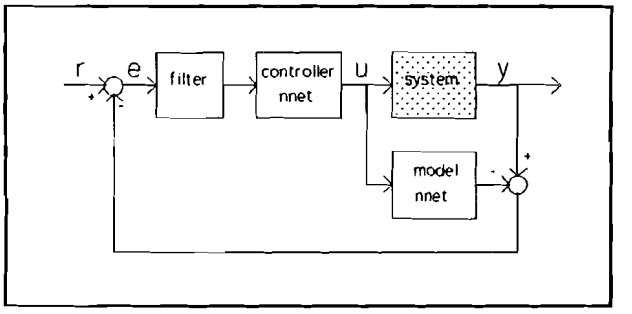


Fig.2.1 Internal Model Control

Two neural networks have to be found, one describing the plant and one describing the controller. If the model is a perfect representation of the plant and both the plant and the controller are stable then the closed loop system is stable. The control will be perfect (r=y) if the controller is the inverse of the model describing the plant, and the closed loop system is stable. The system must be invertable (for linear systems it may not contain unstable zero dynamics). Exact setpoint following will be possible despite unmeasured disturbances. A two step procedure is used to find both networks. The filter is introduced to reduce the gain of the feedback system to increase robustness, because the model obtained will never be perfect.

In step one (fig. 2.2) the model neural network is trained on the error signal e_1, the difference between the plant output and the model neural network output. In the second

step (fig. 2.3) the controller is trained on e_2, the difference between the controller neural network input and the model neural network output. During execution both networks can be trained further on line.

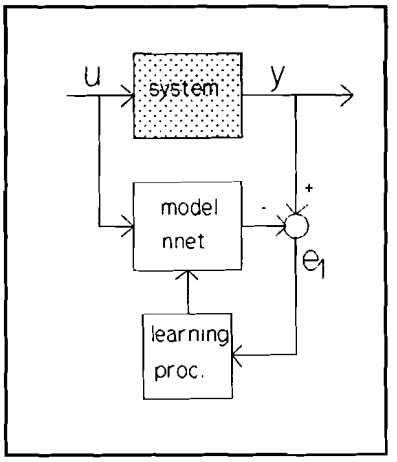


Fig.2.2 Step 1
Model training

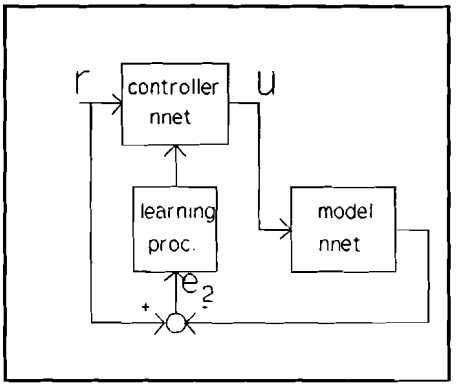


Fig.2.3 step 2
Controller training

2.1.2 Model Reference Indirect Adaptive Control

Narendra [10] uses indirect adaptive control (fig. 2.4). The designer is supposed to be sufficiently familiar with the plant under consideration to prescribe a reference model which specifies the desired behaviour of the system. The model is perceived in the same way as the one for internal model control. The controller will have to generate a plant input that satisfies some criterion for the output. It will be trained on e_1, the difference between the plant output and the reference model output.

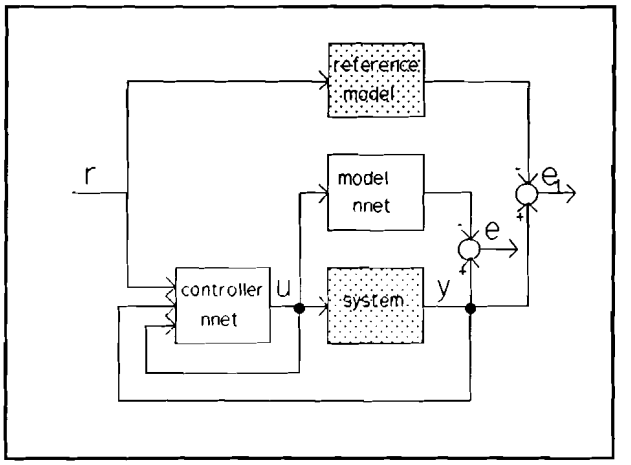


Fig.2.4 Indirect Adaptive Control

2.2 Process Description of the Overhead Crane

The derivation of formulas and the Simulink simulation model are also given by Schreppers[15]. Simulink is a simulation program working with Matlab.

A two dimensional space is defined in which the load can be moved. The x-axis is defined along the rail, the y-axis perpendicular to it. The position of the load is specified in x_L and y_L. These two parameters can be calculated from the three variables that are

returned by the system; the length of the cable L, the position of the trolley x_t and the angle θ between the normal of the rail and the cable by: $x_L=x_t+L\sin(\theta)$; $y_L=L\cos(\theta)$.

The physical equations of the overhead crane could be obtained by defining a pole coordinate system that moves with the trolley along the x-axis. This is causing an extra acceleration (like in a rotating carroussel) according to the x,y system. For the two accelerations in the tangential direction and in the radial direction this will be $a_L=\ddot{L}-L\dot{\theta}^2$; $a_\phi=L\ddot{\theta}+2\dot{L}\dot{\theta}$. Now with the force balances for the load and for the trolley three system equations are derived.

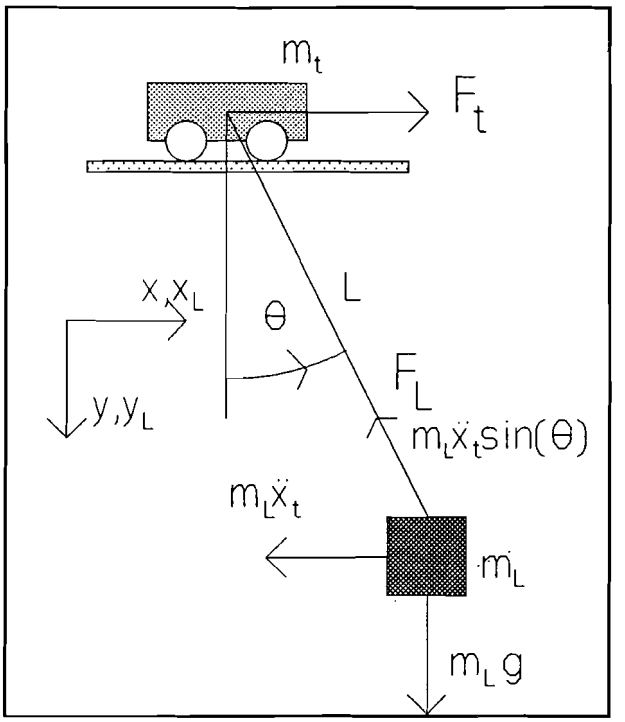


Fig.2.5 Schematic Representation of the Crane

F_t is the x-directional force on the trolley; F_h is the hoisting force; m_t is the mass of the trolley; m_L is the mass of the load. The d's represent damping forces that work opposite to the moving direction. The system equations are:

$$\begin{aligned} m_t\ddot{x}_t &= F_t+F_h\sin(\theta)-d_t\dot{x}_t \\ m_L L\ddot{\theta} &= -m_L g\sin(\theta)-m_L\ddot{x}_t\cos(\theta)-2m_L\dot{L}\dot{\theta}-d_\theta L\dot{\theta} \\ m_L\ddot{L} &= m_L g\cos(\theta)-F_h+m_L L\dot{\theta}^2-m_L\ddot{x}_t\sin(\theta)-d_L\dot{L} \end{aligned} \qquad (2.1)$$

Some important simplifications were stated. The rope is always stiff, the damping forces work linearly with the moving speed, the sensors and actuators of the system are ideal.

Not all of the constants in (2.1) are certain because the real system is not yet available. We want the system to be able to handle all loadmasses within prescribed bounds. As this mass will not change during the trajectory a constant is chosen for m_L. The constants of the system are chosen to be:

$$\begin{aligned} m_t &= 3.5\ kg & d_t &= 0.10\ kg/s \\ m_L &= 1.5\ kg & d_\theta &= 0.01\ kg/s \\ g &= 9.8\ m/s^2 & d_L &= 10.0\ kg/s \end{aligned} \qquad (2.2)$$

2.3 Prestabilization

The system described by the three formulas (2.1) contains one badly damped term in the swing of the load and two integration terms, one for x_t and one for L. These properties

will make this system very hard to identify; it will be difficult to make a dataset using white noise random input signals with the outputs in the operating range. A badly damped term is very difficult to identify as it has a very long response so a very long dataset will be necessary. To overcome these three problems the system is prestabilized by a linear controller. The prestabilized system will be more linear than the original system.

2.3.1 Linearization of the Process

For implementation of an LQR controller, the system must be described by linear equations so the process has to be linearized. The K_{LQR}- factors obtained will be implemented directly on the nonlinear system. The linearization takes place around a working point. The variables in the equations are replaced by

$x_t=x_{t_0}+\Delta x_t;\ \theta=\theta_0+\Delta\theta;\ L=L_0+\Delta L;\ \dot{x}_t=\dot{x}_{t_0}+\Delta\dot{x}_t;\ \dot{\theta}=\dot{\theta}_0+\Delta\dot{\theta};\ \dot{L}=\dot{L}_0+\Delta\dot{L}$

the working point is chosen to be $x_{t_0}=0;\ \dot{x}_{t_0}=0;\ \theta_0=0;\ \dot{\theta}_0=0;\ L_0=1;\ \dot{L}_0=0$.

These new variables are substituted in the system equations (2.1). The following linearization rules are used:

$$\cos(x)\approx 1 \quad \textit{if } x\approx 0;\quad \sin(x)\approx x \quad \textit{if } x\approx 0;\quad \textit{write: } \frac{d(c+\Delta x)}{dt}=\Delta\dot{x};$$
$$\textit{neglect } 2^{nd} \textit{ order differences: } \Delta x\Delta y\approx 0;\ (\Delta x)^2\approx 0 \tag{2.3}$$

The linearized equations describing the system are

$$\begin{aligned} m_t\cdot\Delta\ddot{x}_t &= F_t+m_Lg\theta-d_t\cdot\Delta\dot{x}_t \\ m_L\cdot\Delta\ddot{\theta} &= -\frac{m_Lg}{L_0}\cdot\Delta\theta-\frac{m_L}{L_0}\left(\frac{F_t}{m_t}+\frac{m_Lg}{m_t}\cdot\Delta\theta-\frac{d_t}{m_t}\cdot\Delta\dot{x}_t\right)-d_\theta\cdot\Delta\dot{\theta} \\ m_L\cdot\Delta\ddot{L} &= -F_h^*-d_L\cdot\Delta\dot{L} \qquad \textit{where } F_h^*=F_h-m_Lg \end{aligned} \tag{2.4}$$

We see that there is no coupling between the third and the other two equations (2.4), so the θ and the x-position of the trolley are not dependent on the length of the cable and vice versa. The nonlinearities of the system are apparent only if the angle θ is large. So we can only show the improvement by using a nonlinear method over linear methods if this large θ is happening.

2.3.2 LQR Controller

With the linear feedback parameters a new state space description of the system is specified by:

$$\begin{aligned} \underline{\dot{x}} &= A\underline{x}+B\underline{u} \quad &=> \quad \underline{\dot{x}} &= (A-BK_{LQR})\underline{x}+B'\begin{pmatrix}x_r & y_r\end{pmatrix}^T \\ \underline{y} &= C\underline{x} \quad &=> \quad \underline{y} &= C\underline{x} \end{aligned} \tag{2.5}$$

$$B':=BK_{LQR}\begin{bmatrix}1&0&0&0&0&0\\0&0&0&0&1&0\end{bmatrix}^T$$

Here K_{LQR} is a matrix describing the LQR factors. For the linearized system the state space matrices are specified by:

$$\underline{x} = \left(x_t \; \dot{x}_t \; \theta \; \dot{\theta} \; L \; \dot{L} \right)^T \qquad \underline{u} = \left(F_t \; F_h \right)^T$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{d_t}{m_t} & \frac{m_L g}{m_t} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{d_t}{m_t L_0} & -\frac{g}{L_0}(1+\frac{m_L}{m_t}) & -\frac{d_\theta}{m_L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -\frac{d_L}{m_L} \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ \frac{1}{m_t} & 0 \\ 0 & 0 \\ -\frac{1}{m_t L_0} & 0 \\ 0 & 0 \\ 0 & -\frac{1}{m_L} \end{bmatrix} \tag{2.6}$$

{A,B} is controllable. The LQR routine will minimize this performance function:

$$J_{LQR} = \int_{t_0}^{\infty} \frac{1}{2}(x^T Q x + u^T R u)\, dt \tag{2.7}$$

For Q the usual C^TC doesn't suffice because then the system still has a badly damped θ. So for the term in Q for feedback on θ the factor 10 is chosen. The R matrix determines the maximum input signals. These depend on the maximum allowable actuator signals, the maximum torques of the motors. The Q and R matrices are:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{2.8}$$

The K_{LQR}-factors were found by solving a Riccati Equation resulting in:

$$K_{LQR} = \begin{bmatrix} 10.0000 & 11.7675 & -19.0059 & -2.5225 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & -10.0000 & -1.3952 \end{bmatrix} \tag{2.9}$$

The pole zero plots of the MIMO linear system in open loop and closed loop (with the

linear feedback) are plotted in fig. 2.6.

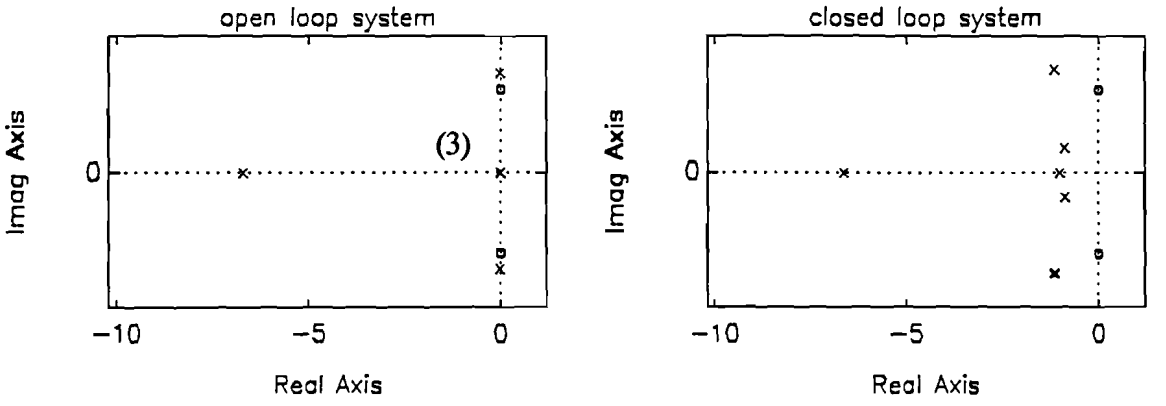


Fig.2.6 Pole Zero Plots Linear System; Open Loop and with K_{LQR}

2.3.3 Loose Feedback

The system with the LQR controller has very short impulse responses, so it is controlled very tightly. What we would like to have is some freedom of excitation for the neural network controller so we release the feedback rates to K_{LOOSE}. The pole zero plot of the closed loop linear system with K_{LOOSE} is plotted in figure 2.7. The values of K_{LOOSE} are:

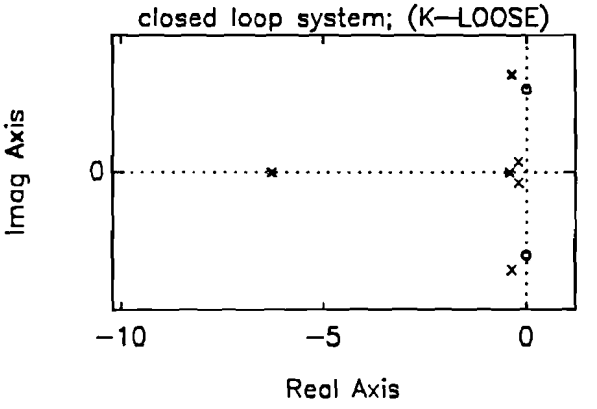


Fig.2.7 PZ Plot Linear System with K_{LOOSE}

$$K_{LOOSE} = \begin{bmatrix} 1.0 & 2.0 & 0.0 & -2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -4.0 & 0.0 \end{bmatrix} \quad (2.10)$$

The outputs of the system that we really want to control are not the states but the two outputs x_L and y_L that specify the trajectory. The new feedback variables are applied on x_L and y_L in stead of x_t and L, and the four other states.

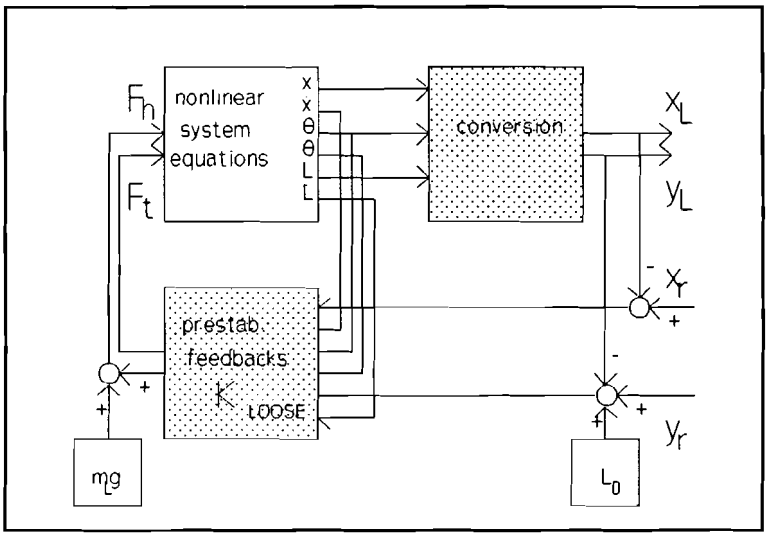


Fig.2.8 Set up for Identification

The responses of x_L and y_L to simultaneous impulses on x_r and y_r are plotted both for the nonlinear system with the LQR feedback and the nonlinear system with the loose feedback in fig.2.9.

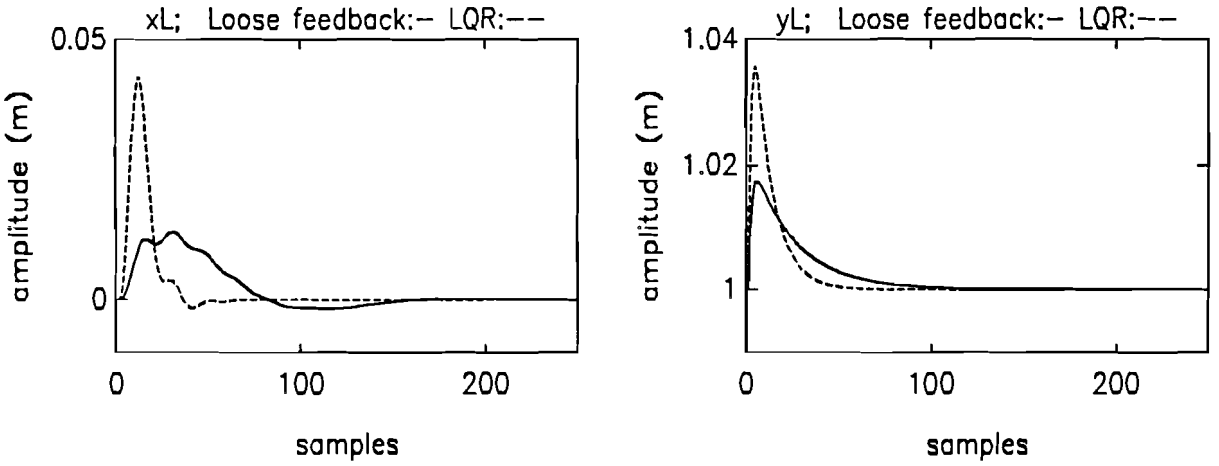


Fig.2.9 Nonlinear System Impulse Responses; K_{LQR} and K_{LOOSE} (sampling freq. 10 Hz)

The configuration that will be used for control is drawn in figure 2.8. This system will be used in the identification experiments. It has two inputs x_r and y_r and two outputs x_L and y_L that are not measured directly, but by means of three sensors for x_t, L and θ (x_L and y_L are calculated in the block 'conversion' in fig.2.8). Be aware that for a large cable length a small change in θ results in a big change in x_L, so this will be less accurately measured. The system with the feedback loop is implemented in the simulation program Simulink.

A simulation has some advantages over a real system. We are able to work with deterministic signals, noise can be added. The calculation is faster than real time. We're not bound to a physical implementation. As we want the crane to handle different load masses we could use the mass as one of the random input signals in data retrieving. Two disadvantages are that we don't know whether the model is a good approximation of the real system and how high the real noise levels of the system are.

2.6 Data Acquisition

For identification of the system we will produce a data set in which sufficient information about the system is stored. For the input signals pseudo random binary noise sequences that are often used in linear system identification are not proper, because it has only two amplitude levels; also the amplitudes of the input signals determine the amount of nonlinearity. A uniformly distributed white noise input signal was chosen, so all the possible input situations have the same probability to occur. The maximum amplitudes of the white noise signals are dependent on the maximum actuator signals. These were chosen so that the whole operating range of the output space is covered and the angle θ is as big as possible (up to ± 0.5 rad). Only for a big angle θ big nonlinearities occur, so this is the way to show improvement over linear methods.

For complete reconstruction the sampling frequency should be at least twice the highest system frequency. Usually at least ten times this frequency is chosen to allow data preprocessing. The highest mode of the system is the swing frequency of the load which is dependent on the length of the cable. In the working point it is determined from the impulse response to be 0.6 Hz. Theoretically this pendulum frequency can be calculated by $f_{pend} = \frac{1}{2\pi}\sqrt{g/L}$, so if the length of the cable is chosen in the range $0.1\text{m} < L < 1.9\text{m}$ it should be less then 2 Hz. Also the sampling frequency should not be too high because systems containing longer impulse responses are difficult to identify. The sampling frequency of 10 Hz was chosen. Now the impulse response lengths for both outputs are about 200 samples. In linear system identification the length of the data set is usually chosen about ten times the maximum impulse response length (it depends on the disturbance level that is unknown). In nonlinear system identification a longer dataset might be necessary to map all the nonlinearities. We chose for a dataset of 5000 samples, which is about 25 times the maximum measured impulse response length. Actually it is not certain that this is long enough but a longer dataset would lead to problems in the processing software.

Simulink uses numerical integration of sets of ordinary differential equations. The user is allowed to choose amongst different integration algorithms. The very accurate Adams method was used which is said to be used for smooth and nonlinear systems with no widely varying time constants. The user specifies the sampling time of the system, in this case it is 0.1 s. To approach the continuous system an internal time step (that is much smaller than the sampling time) is used in the integrations. The internal time step is automatically adjusted to ensure that the signals are calculated accurately.

2.5 Data Preprocessing

There is a delay of one sample for both outputs of the system because of the calculations; the output is shifted one sampling time step.

It is allowed to use the linear transformations, scaling and offset correction on the input and output signals if the proper inverse transformations are used on the neural network output signals when applied. The nonlinearities of the system will still be learned by the neural network.

Offset correction takes place to achieve that all the signals have mean amplitude zero. In the system only the output y_L is offset corrected by the linearization point $L_0=1$. The input and output signals are scaled to make sure that all of these have as much weight in the minimization criterion. This is achieved by making sure that:

$$\|x_r\|_2 \approx \|y_r\|_2 \quad \wedge \quad \|x_L\|_2 \approx \|y_L\|_2 \tag{2.11}$$

This is done by multiplying the signal with the smaller norm with a constant factor bigger than one.

CHAPTER 3
Neural Nets

The field of neural networks in control systems has been steadily in progress for the last few years. There are high expectations because neural networks have some very attractive qualities as will be explained in 3.1. In this chapter is explained what kind of neural networks we use. Then the configuration that is used for the identification of the model is shown and some neural network training routines are explained.

3.1 Fundamentals of Neural Nets

The first feature of neural nets (neural networks) is, that bounded sets of non linear functions, if continuous and finite, can be approximated. In most of the more complex real systems and processes nonlinear functions occur. The approximation level is related to the complexity of the network. Also there are algorithms to train the network; a model of the system can be made in black box configuration, using only input and output signals.

In the field of pattern recognition (static neural networks) very good results have been seen. This quality can also be applied in our situation. Think about the fact that external matters like strong wind or the load mass have much influence on the system. Each situation demands its own controller. Now there is a number of controllers amongst which the right one will have to be chosen. The choice of the controller is a pattern recognition problem and can be made by a neural network, see Narendra[12].

By using parallel hardware implementation the calculation can be made very fast. For on line control a very fast calculation is necessary.

In Pijnenburg[13] more basic fundamentals of neural networks are explained. We make use of a completely connected multilayer neural network as drawn in figure 3.1. Each node represents a neuron that performs a (nonlinear) function on the weighted inputs. The variables describing the network are the weights on the interconnections. For the neuron processing function any shape can be chosen (except the output and input nodes that are chosen linear). Frequently used are the sigmoid and the Gaussian function (fig.3.2) . The Gaussian has the advantage that the offset in the saturation region is zero, so this doesn't have to be compensated by another node. The advantage of the sigmoid, in this case $1/(1+\exp(-x))$ (also the very similar tanh function was used), is that it consists of a linear part around zero, a nonlinear part and a saturation. As there is

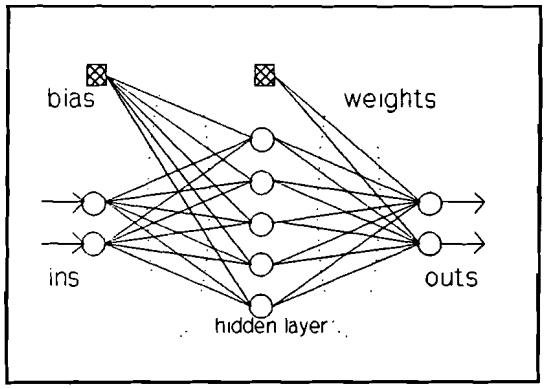


Fig.3.1 Static Neural Net

a big linear part in the system we chose for the sigmoid function. We expect that this facilitates the approximation of linear systems. A neuron is said to be in its linear region if the sigmoid is operating in its linear part.

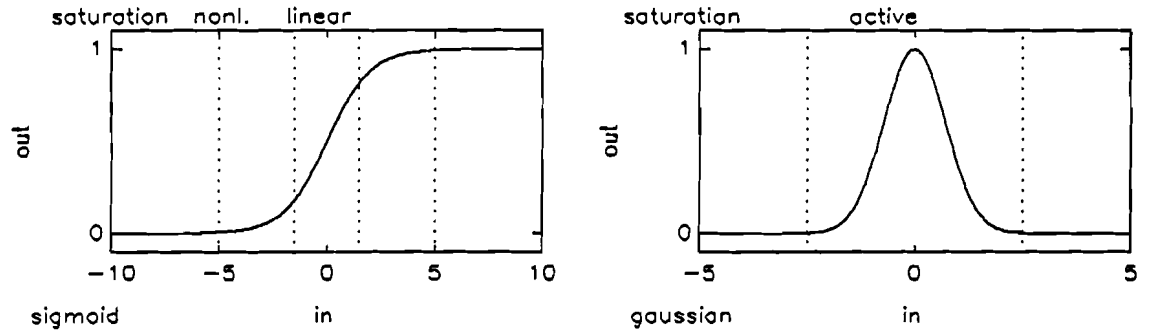


Fig.3.2 Processing Functions

Any static function can be approximated by this network up to some accuracy related to the size of the network. There are freedoms in the number of hidden layers and the number of nodes per hidden layer.

We represent a Neural Net by $NN_{i_0\, i_1 \ldots i_L}$ where L is the number of layers, i_0 is the number of inputs, i_1 is the number of nodes for the first hidden layer, i_{L-1} for the last hidden layer and i_L is the number of outputs. $NN_{1\,2\,3}$ has one input, one hidden layer with 2 nodes and 3 outputs.

3.2 Dynamic Neural Networks

To bring in dynamics into the static neural network shown in figure 3.1 some delayed inputs and outputs are fed back to the network (for linear systems the number of delayed outputs is proportional to the number of states of the system). This is the ARMA dynamic neural network. It is plotted in figure 3.3. In stead of using basic shifts also other filters (of McMillan degree one) can be used on the delayed inputs. In fact the shift is a special case. In fig. 3.3 Δ is the shift operator σ or z^{-1}.

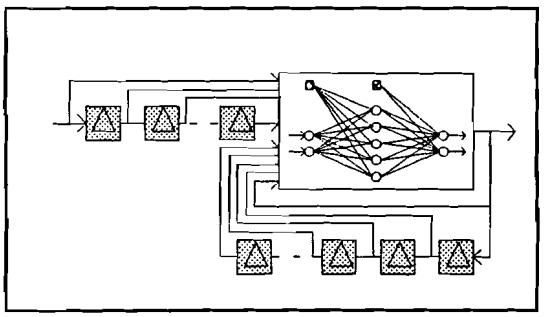


Fig.3.3 Dynamic Neural Net

3.2.1 Delta representation

Besides the shift operator three other possible delta filters are

$$(1):\ \Delta_1=\frac{z-1}{T};\quad (2):\ \Delta_2=\frac{z-1}{zT};\quad (3):\ \Delta_3=\frac{z-\alpha}{\alpha z-1} \tag{3.1}$$

T is the sampling time. The first filter is non causal so it cannot be implemented here. The second filter creates an approximation of the derivatives of the input and output signals:

$$\Delta_2 y(k)=\frac{z-1}{Tz}y(k)=\frac{1}{T}(y(k)-y(k-1))\simeq \dot{y} \tag{3.2}$$

$$\Delta_2^2 y(k)=\frac{1}{T}(\Delta_2 y(k)-\Delta_2 y(k-1))\simeq \ddot{y} \tag{3.3}$$

The reason of implementing this filter is that the derivatives of the outputs contain more evidently the information about the system than the shifted outputs, especially when there is small change in the outputs during successive time steps.

A problem in the use of these filters is that $\|\Delta_2 y\| \neq \|y\|$. If for example for one of the outputs $\|\Delta_2 y\| \gg \|y\|$, say ten times as big, in the training much more weight is placed on the derivatives of this signal, because of the higher amplitude of the derivatives. This will make it very hard to identify the system, especially when higher derivatives are also used as inputs (the fourth shifted output signal will have an amplitude that is 10000 times bigger than the output). This effect can be counteracted by using a constant factor other than $1/T$ in the delta function. The neural network will take care of the effects of this factor.

Another way to overcome this problem is to use the fourth filter for which goes $|\Delta_3(e^{i\omega})|=1 \quad \forall\ \omega\in[-\pi,\pi]$. So it is an all pass filter. Actually a special case of this filter is the time shift.

3.3 Training Neural Networks

For the identification of the neural model the network is trained in the output error configuration plotted in figure 3.4. With this configuration a model will be found that only uses the inputs of the system to approach the system outputs. The performance J is minimized (in the algorithms just the numerator in 3.4. is minimized).

$$J = \frac{\sum_{k=1}^{N}\sum_{i=1}^{p}\left[y_{i,model}(k)-y_{i,process}(k)\right]^2}{\sum_{k=1}^{N}\sum_{i=1}^{p}\left[y_{i,process}(k)\right]^2} \tag{3.4}$$

N is the number of samples on which this criterion is minimized (typically 5000 samples)

and p is the number of outputs of the system. We will have to find the right weights describing the neural network in an m-dimensional space, where m is the number of weights. Because of the nonlinearities this is a nonconvex problem. There is no guarantee for success to find the global minimum. Besides the global minima some local minima can exist and flat regions and saddle points appear. Theoretically a neural network with more weights can approximate the system better but the more weights the slower is the convergence rate and the more difficult it is to find the minimum. With these two facts the optimum network size has to be found empirically.

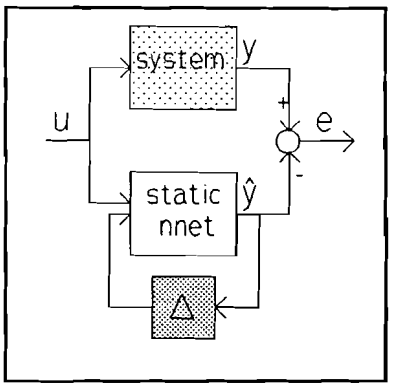


Fig.3.4. Output Error Method

There is no a priori indication of the value of the performance in the global minimum. So if the neural network is in a minimum it is hard to say whether it is a global one. We can only compare with the performance of models obtained by linear identification experiments.

3.3.1 Gradient Methods

Some minimization routines use the gradients dJ/dw of the performance to the weights w. It is not always possible to escape from a local minimum when a gradient method is used. The gradients can be determined in two ways, analytically and numerically. In the numerical calculation for each weight the performance has to be recalculated in a point close to the current point. The analytical method is explained by Narendra[10] and is used to save flops. Moonen[7] wrote a C-program for analytical determination of gradients for the ARMA representation neural network. The gradients are calculated according to the formula (3.5) where the output of the neural network is represented as follows:

$$\hat{y}(k) = f(u(k),u(k-1),u(k-2),\ldots,\hat{y}(k-1),\hat{y}(k-2),\ldots,w), \quad k=1,2,\ \ldots$$

$$\frac{dJ}{dw_j} = \frac{1}{2}\sum_{i=1}^{p}\sum_{k=1}^{N} e_i(k)\frac{d\hat{y}_i}{dw_j}(k) \quad \text{where} \quad e_i(k) = y_i(k)-\hat{y}_i(k) \tag{3.5}$$

$$\frac{d\hat{y}_i}{dw_j}(k) = \frac{\delta\hat{y}_i(k)}{\delta w_j} + \sum_{n=1}^{p}\sum_{t=1}^{ndo}\frac{\delta\hat{y}_i(k)}{\delta\hat{y}_n(k-t)}\frac{d\hat{y}_n}{dw_j}(k-t) \tag{3.6}$$

j is the number of weights and ndo the number of delayed outputs fed back.

This is the dynamic backpropagation algorithm, if the second term of formula 3.6 is neglected it is called static back propagation. The neglection is allowed if there are no lightly damped terms to be identified.

In the second Delta representation $(3,1:\Delta_2)$ neural network the dynamic term is changed. Writing Δ for Δ_2:

$$\hat{y}(k) = f(u(k),\Delta u(k),\Delta^2 u(k),\ldots,\hat{y}(k-1),\Delta\hat{y}(k-1),\Delta^2\hat{y}(k-1),\ldots,w), \quad k=1,2,\ \ldots$$

$$\frac{d\hat{y}_i}{dw_j}(k) = \frac{\delta\hat{y}_i(k)}{\delta w_j} + \frac{\delta\hat{y}_i(k)}{\delta\Delta\hat{y}_i(k-1)}\frac{d\Delta\hat{y}_i}{dw_j}(k-1) + \ldots + \frac{\delta\hat{y}_i(k)}{\delta\Delta^{ndo}\hat{y}_i(k-1)}\frac{d\Delta^{ndo}\hat{y}_i}{dw_j}(k-1) \tag{3.7}$$

So the analytical calculation takes extra flops.

$$\frac{d\Delta^n\hat{y}_i}{dw_j}(k-1) = \frac{d}{dw_j}\frac{\Delta^{n-1}\hat{y}_i(k-1)-\Delta^{n-2}\hat{y}_i(k-2)}{T} = \frac{1}{T}\left[\frac{d\Delta^{n-1}\hat{y}_i}{dw_j}(k-1)-\frac{d\Delta^{n-1}\hat{y}_i}{dw_j}(k-2)\right] \tag{3.8}$$

The implementation of this in the software is very difficult (time consuming) and relatively less time profit can be achieved in the calculation so we decided to calculate the gradients numerically.

Steepest Descent

The basic gradient optimization method is steepest descent, that is used in pattern recognition problems. The weights are adapted according to

$$w(k+1)=w(k)-\alpha\nabla_w J(w) \tag{3.9}$$

w is the vector of weights. ∇_wJ(w) is the gradient of the performance to the weights. A proper learning velocity factor α has to be chosen; with a small α the convergency rate is very small; with a large α there is less guarantee of convergence. The convergence rate can be increased by making α adaptive. Increase α after a number of successful iterations. Decrease α when the performance is worsening.

In Pijnenburg[13], Jacobs[5] and Qiu[14] other methods to speed up the steepest descent method are mentioned.

Quasi Newton

When we compare the gradient methods the steepest descent converges linearly, the Quasi Newton converges quadratically. Quasi Newton can be used only if the values of the parameters are close to the minimum. M is a positive definite approximation of the Hessians, H in (3.10). M is obtained by updating the initial identity matrix.

$$w(k+1) = w(k) - \alpha M\nabla_w J(w); \qquad H = \left[\frac{\delta^2 J(w)}{\delta w \delta w^T}\right] \tag{3.10}$$

3.3.2 Nongradient methods

Stochastic search (Telkamp[16]) is an algorithm that determines a new point stochastically. For each direction the mean of the random step is related to the success of the search in that direction. The convergence rate of stochastic search is very small but it can escape from local minima. In the experiments the results of stochastic search are used as the initial weights for the gradient optimization programs.

$$
\begin{aligned}
w(k+1) &= w(k)+P_k; \qquad P_k \textit{ random standard deviation } \sigma(k) \\
\sigma(k+1) &= \alpha\sigma(k)+(1-\alpha)\frac{a(k)}{P_a}\beta|P_k| \\
&\textit{if } J(k+1)<J(k) \textit{ then } a(k)=1 \textit{ else } a(k)=0;\ 0<\alpha<1
\end{aligned}
\tag{3.11}
$$

At each iteration a random vector P_k with standard deviation σ is created. In each iteration σ is adapted, decreased when not successive a(k)=0, increased when successive a(k)=1. P_a is a chosen constant. β is determined by the probability function. For a complete description of this algorithm see Telkamp[16].

3.4 Minimization in Practice

In simple convex optimization problems the gradient algorithms have shown to be very appropriate. Problems arise when there are flat regions and curved valleys. For the minimization of our neural nets we start with small random initials, so all the nodes are in the linear region. The convergence rate is decreased when some of the nodes are in the saturation region. During the optimization there appears a big difference in the values of the gradients of the network. Sometimes the biggest gradient is 1e10 times bigger than the smallest gradient. Because of the small gradients the steps are small and the convergency rate decreases. When also the derivatives of the gradients are small the Hessians become ill-conditioned (close to singular) and the optimization using the Quasi Newton method cannot be continued.

The famous "banana function"(3.12) shows that already in a two-dimensional error surface it can be very hard to find the global minimum using the gradient methods.

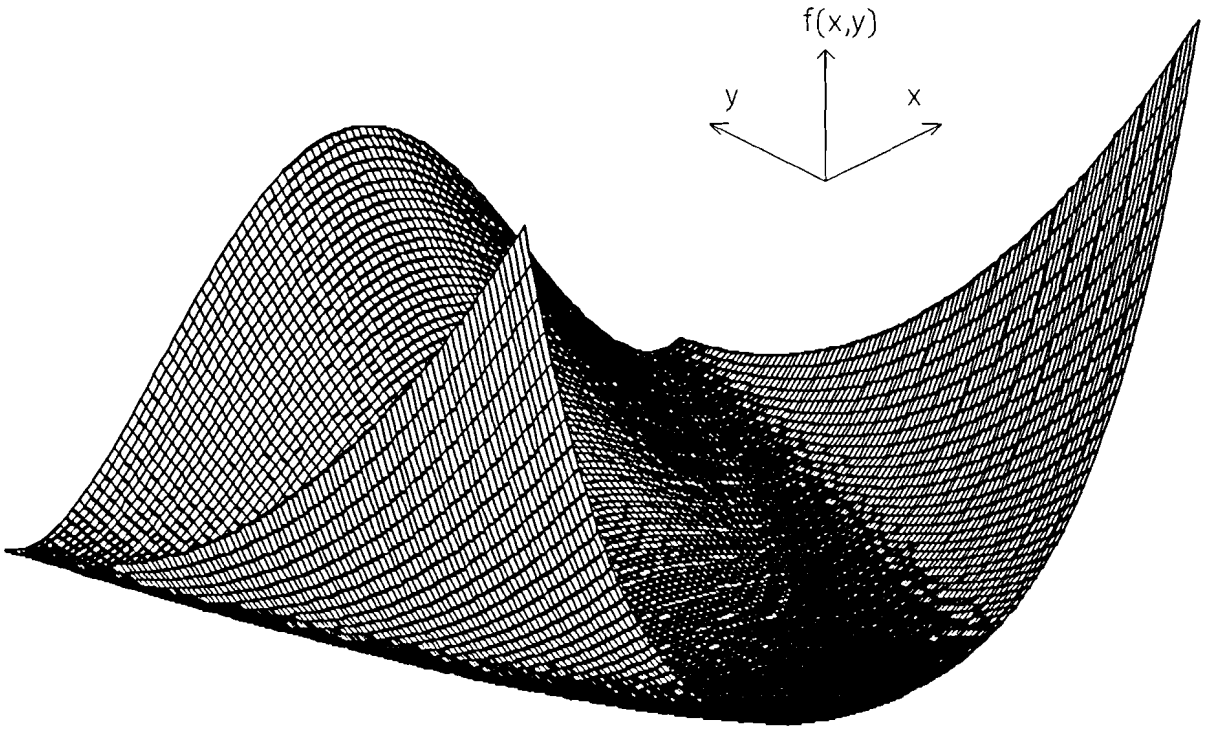


Fig. 3.5 The Banana Function

$$f(x,y) = 100(y-x^2)^2-(1-y)^2 \qquad (3.12)$$

plotted: $-2<x<2 \quad 0<y<1$

In the minimization routine much more weight is put on the first term in formula (3.12). It is no problem to reach the valley. Inside this valley, however, it takes a lot of iterations to get closer to the minimum, because the gradients and the derivatives of the gradients are very small.

In neural network training the dimension of the parameter space is the number of weights (this can easily be 100 or more). Another property of neural networks that makes it more difficult to train the network is the redundancy.

A procedure was set up in which only the weights of the neural network belonging to the biggest gradients were allowed to change during optimization. Only in these few directions was searched. As the gradients in the output layer are often big and this is a linear layer, this tends to a more linear optimization problem. The problem is that after a few iterations the gradients of the other weights are getting bigger. This procedure has to be repeated many times to prove success.

Another procedure is indicated in (3.13). An extra term is introduced in the performance

function J (3.4) leading to J_{new}, to keep the weights small. With small weights the nodes are in the linear range and the convergence rate is higher. t_k is chosen close to one in the beginning of the minimization and is slowly decreased to zero. Finally the performance J_{new} equals the original performance. In section 4.5 it is shown that this procedure proves to be successful.

$$J_{new} = (1-t_k)J + t_k * \frac{1}{2n} w^T w$$
$$\lim_{k \to \infty} t_k = 0; \quad e.g.\ t_k = (0.9)^k \qquad (3.13)$$
$$1 > t_k > 0 \quad n := dim(w)$$

The stochastic search algorithm is able to escape from local minima but practically there is no guarantee that it will succeed. To increase this possibility the learning velocity factor α has to be chosen close to one. Now after a successful iteration the standard deviation of the random vector is increased and it takes a lot of iterations before there is a substantial possibility of a new successful iteration to occur. In practice the convergence rate of stochastic search is too small to find a global minimum of a complex neural network.

As the size of the network is closely related to the convergence rate, the networks are kept as small as possible.

The training of the neural network is started with random search taking small random initial weights. After this the steepest descent is applied to get close to the minimum. Then this minimum is found with Quasi Newton.

Experiment

In this experiment it will be shown that for a simple example there is a minimum bound to the number of hidden layers of the neural network.
The two input one output exclusive or function exor is simulated: y(0,0)=0; y(0,1)=1; y(1,0)=1; y(1,1)=0. The processing function of the neural network is the step function (fig. 3.6).

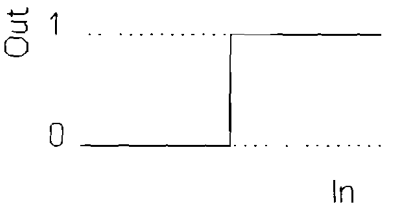


Fig.3.6. Stepfunction

With the neural network $NN_{2\,1}$ (no hidden layer) a decision boundary between y=0 and y=1 in the two dimensional output space is a straight line. This can be shown because with the three weights of the network, the switching of y occurs at:

$$w_0 Bias_0 + w_1 x_1 + w_2 x_2 = 0 \quad with\ inputs\ x_1, x_2$$

With this network the exclusive or function cannot be simulated as shown in fig. 3.7.

With the neural network $NN_{2\,2\,1}$ with nine weights the boundary is determined by two

straight lines:

$$w_0 Bias_0 + w_1 x_1 + w_2 x_2 = 0; \quad w_3 Bias_0 + w_4 x_1 + w_5 x_2 = 0$$

With these two straight lines the exclusive or function can be simulated.

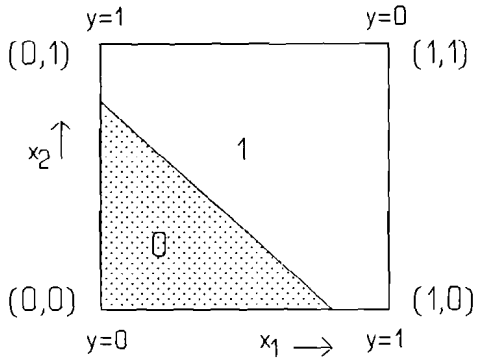


Fig.3.7. No hidden layer

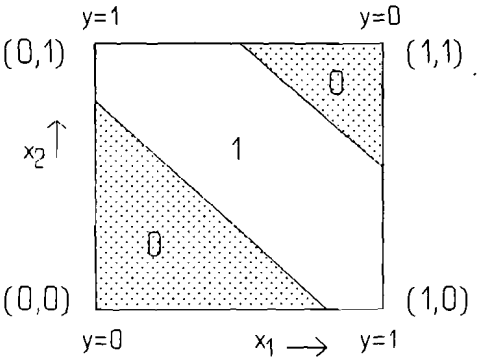


Fig.3.8. 1 hidden layer

It can be shown (see Minsky[6]) that with two or more hidden layers arbitrary regions can be partitioned, the complexity of the regions is determined by the number of nodes. Of course this goes only for this simple two-dimensional output space.

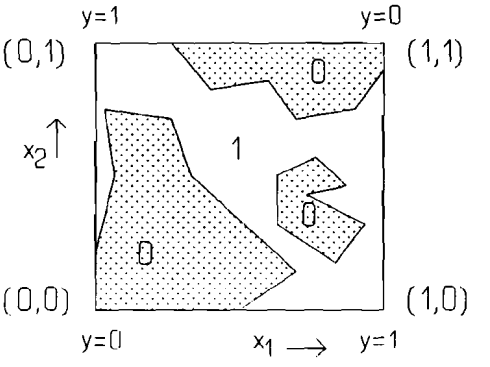


Fig.3.9. 2 hidden layers

CHAPTER 4
Simulation of the Overhead Crane

In this chapter all the neural network training experiments are the best from a range of experiments with different random initials, different numbers of layers and numbers of nodes per hidden layer. Sometimes different processing functions were tried, but all the shown results are from neural networks with the sigmoid function. What is shown is the validation, that means simulation on another dataset than the one the network was trained on. The minimization of the performance is always started with stochastic search followed by steepest descent to get close to the minimum. When the rate of convergence is getting close to zero we switch to the Quasi Newton method. We only use deterministic signals.

4.1 Experiments Using ARMA Representation Neural Nets

We started with a MIMO simulation experiment of the overhead crane when the software for training the neural network using dynamic backpropagation was available. The software is described by Moonen[8]. It was already tested on a few simple systems.

MIMO simulation experiment

The system has two inputs, x_r and y_r, and two outputs, x_L and y_L, and is prestabilized with the linear feedback K_{LOOSE}. The linear system contains six states $x_t, \dot{x}_t, L, \dot{L}, \theta, \dot{\theta}$, so totally at least six delayed outputs have to be fed back to the network. As the angle θ is mostly seen in the output x_L for this output four delays should be fed back (θ is very small so $\sin(\theta) \simeq \theta$; $\cos(\theta) \simeq 1$). In each hidden layer the number of nodes should at least be equal to the number of outputs of the system to enable it to map all the transfers. This is because all the information has to be transferred through all the layers of the network.

Empirically we found out that with a neural network with two hidden layers a better performance could be attained than with only one. See also the experiment in chapter 3.4. With more than two hidden layers the network is very big and has a lot of redundancy. The convergence rate is small.

The MIMO simulation experiment is started with the Neural network $NN_{16\,8\,8\,2}$, so it has 226 weights. For both inputs three delayed inputs and for each output four delayed outputs are fed back to the network. The network is trained on a dataset of 5000 samples with as inputs uniformly ditributed white noise . The minimization is stopped when no improvement is seen in the performance. The method of only minimizing with the biggest gradients as mentioned in section 3.3.1 is applied.

The performance of the neural network is 1.1% (see formula 3.4) (1.8% for output x_L and 0.4% for y_L). In figure 4.1 the validation is plotted for both outputs and the error signals. The output y_L is better approximated than x_L. The difference between the output

errors becomes clear when we have a look at the simultaneous impulse responses; see figure 4.1 e,f. The pendulum frequency in x_L seems to be missing.

The impulse response x_L of the system contains a low and a high frequency component. The high frequency component is not represented by the neural network. Apparently this component has less effect on the performance of the model than the low frequency one.

With this neural network model the system approximation is not good enough in order to be used for controller design. The performance is bad. The amplitude of the error signal of x_L is up to 10 cm. The high frequency component corresponds to the swinging of the load. Now the swinging of the load is an effect that we want to suppress with the controller. This will not be possible if this frequency is not represented by the model. Because the performance of the Neural Network has a very flat surface as function of the weight parameters w, it could be due to a local minimum. Another strategy will have to be applied to find a better neural network model.

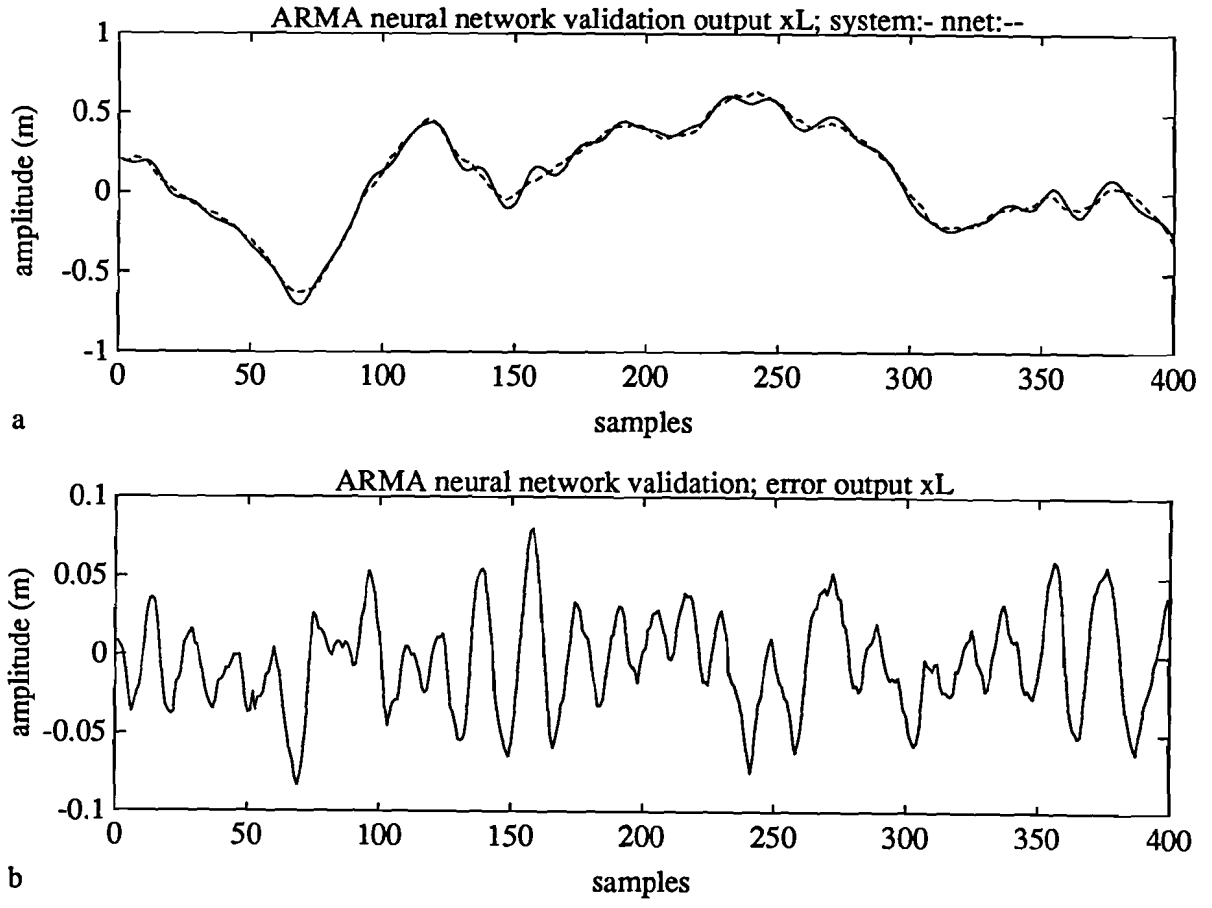
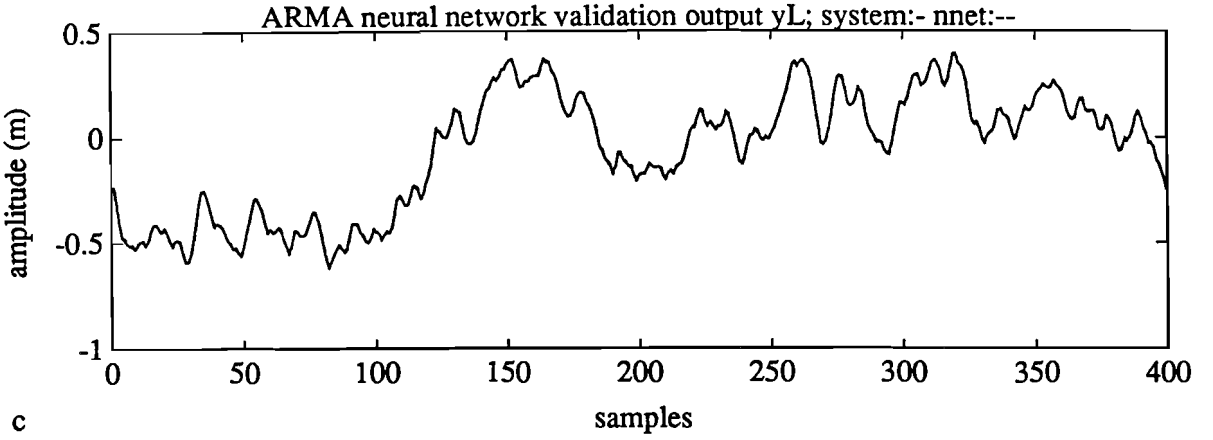
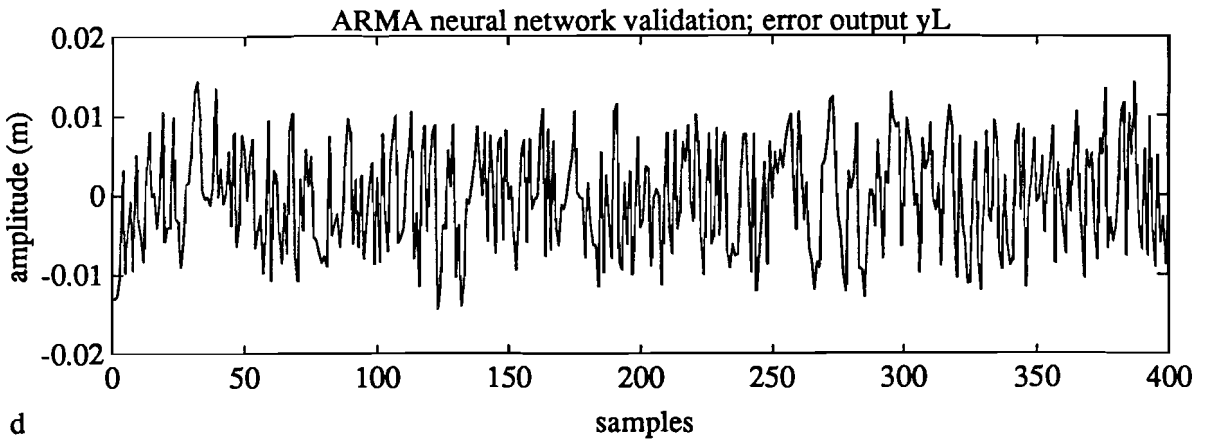


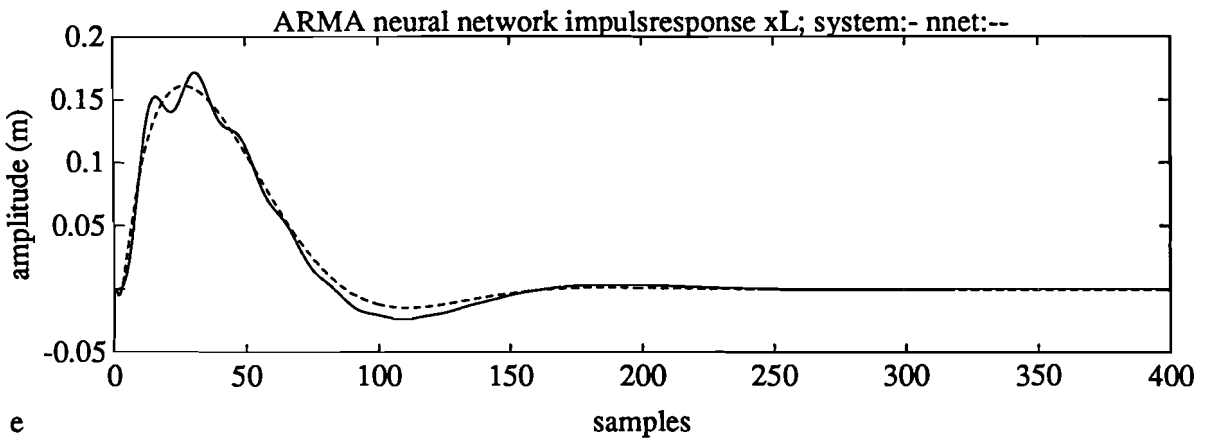
Fig.4.1 MIMO Simulation Experiment ARMA



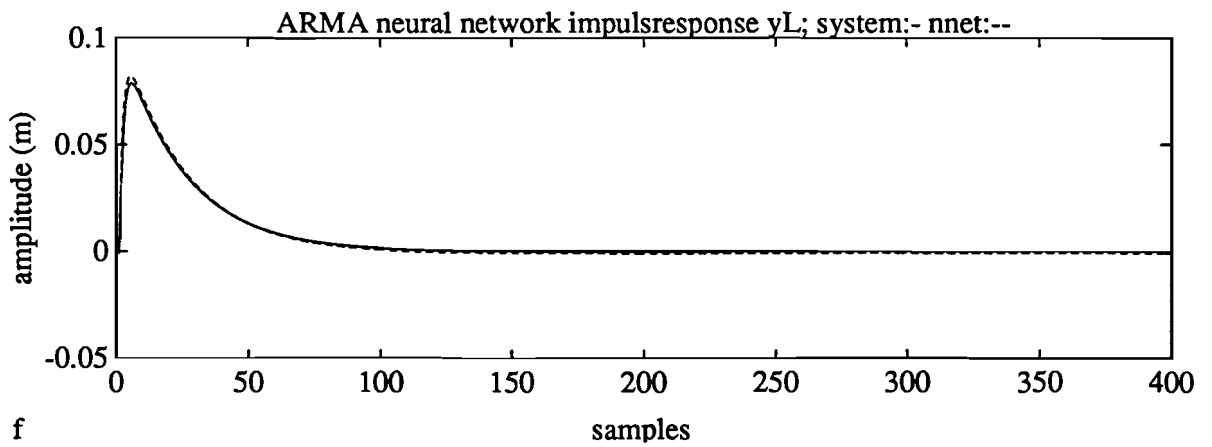
c



d



e



f

Fig.4.1 MIMO Simulation Experiment ARMA (cont.)

4.2 Experiments Using Delta Representation Neural Nets

As explained in section 3.2.1 with the Delta representation method we expect that the training of the neural network will be easier, because the information is more evident.

The delta representation method using the second filter (Δ_2 in 3.2) is tested on a few simple experiments. To our surprise the trained network does not represent the system but generates only constants as outputs. This can be explained by what is stated in section 3.2: there is very much weight on the delayed outputs, that contain less information than the real outputs so a constant value is an important local minimum. The problem can be circumvented by changing the factor 1/T. Another way to counteract this effect is to start with the training of a neural network with just one delayed output. When a quite good performance is attained the other outputs are added to the network taking small random initials for the weights on the extra interconnections. Now the training of the network is continued. This is illustrated in fig. 4.2.

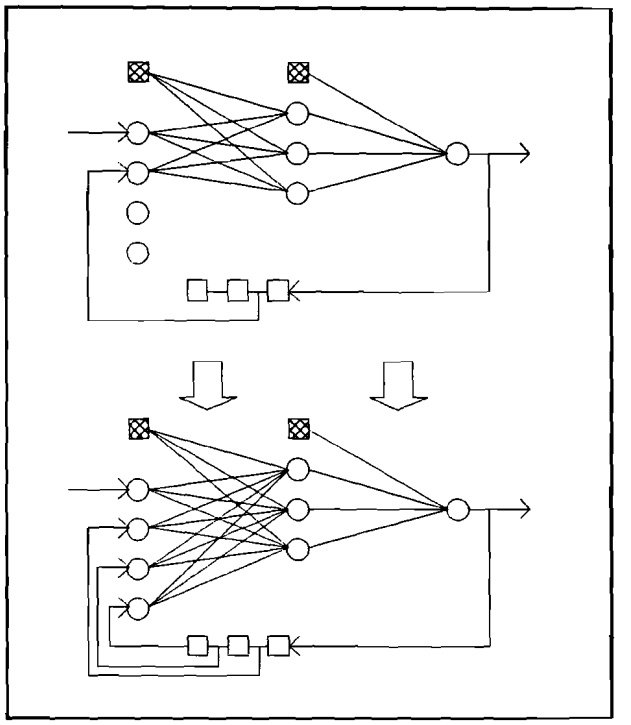


Fig.4.2. Add Extra Delayed Outputs

A MIMO neural network training experiment $NN_{16\,8\,8\,2}$ with inputs x_r, Δx_r, $\Delta^2 x_r$, x_L, Δx_L, $\Delta^2 x_L$, $\Delta^3 x_L$, $\Delta^4 x_L$, y_r, Δy_r, $\Delta^2 y_r$, y_L, Δy_L, $\Delta^2 y_L$, $\Delta^3 y_L$, $\Delta^4 y_L$ is carried out. The same dataset was used as in the ARMA simulation experiment.

In figure 4.3 the results are shown. These are similar to those of the ARMA representation neural network. The performance of the Delta representation model is better: 0.8% (x_L 1.4% y_L .1%). This performance is attained with about half the number of Quasi Newton iterations as used in the ARMA experiment. It was not necessary to continue the minimization of the performance only with the weights belonging to the biggest gradients. However one iteration takes more flops and it took more experiments with different random initials before this neural network was adjusted. By the impulse responses is shown that the frequency of the pendulum is not represented by the model. In the spectrum of the error signal there is a peak for the high frequency term (indicated in figure 4.3 g). The amplitude of this peak is still smaller than the amplitude of the error signal for the low frequency, so when the minimization of the performance is continued still more weight is put on this low frequency.

In all following experiments the ARMA representation is used.

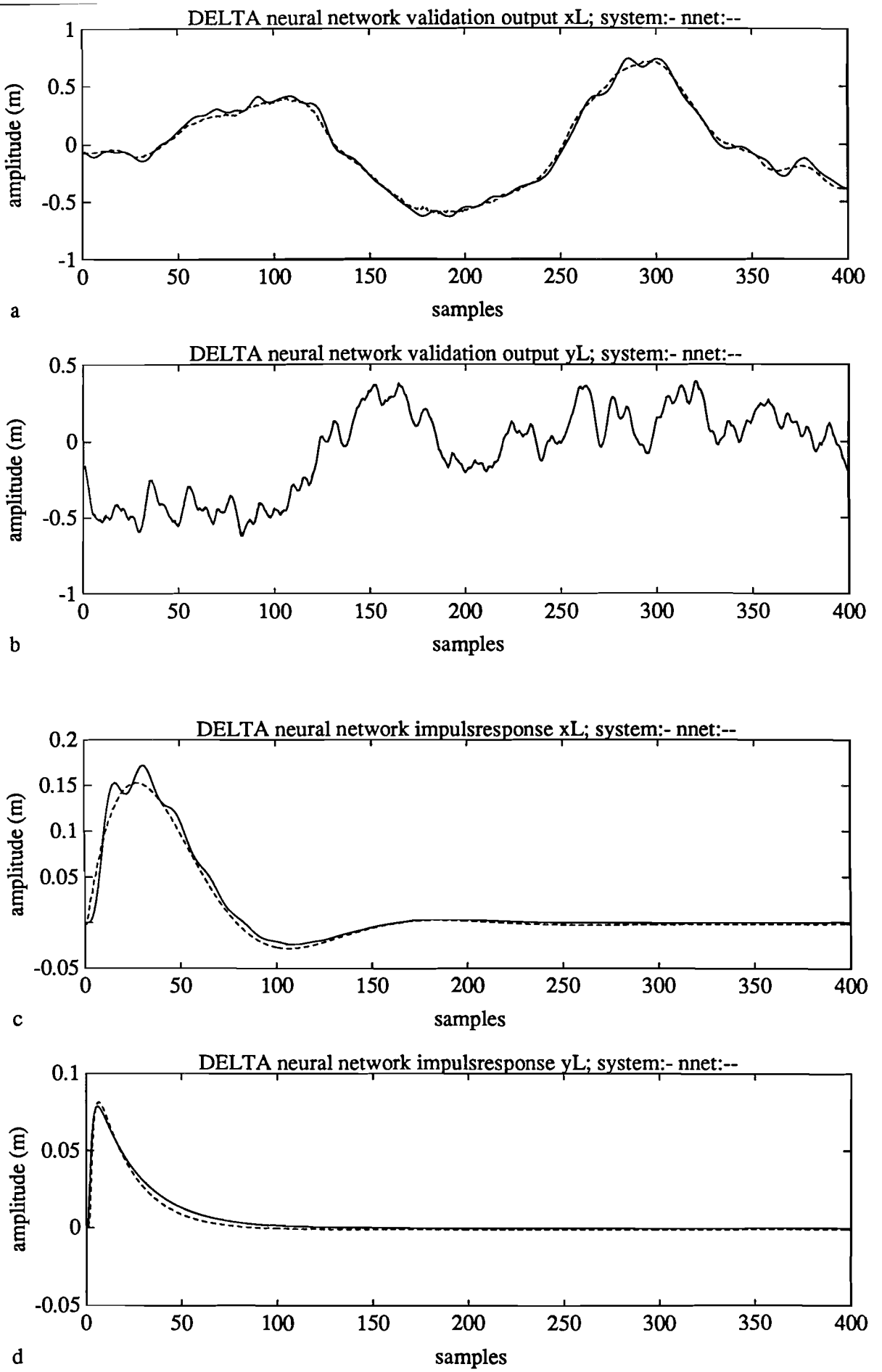
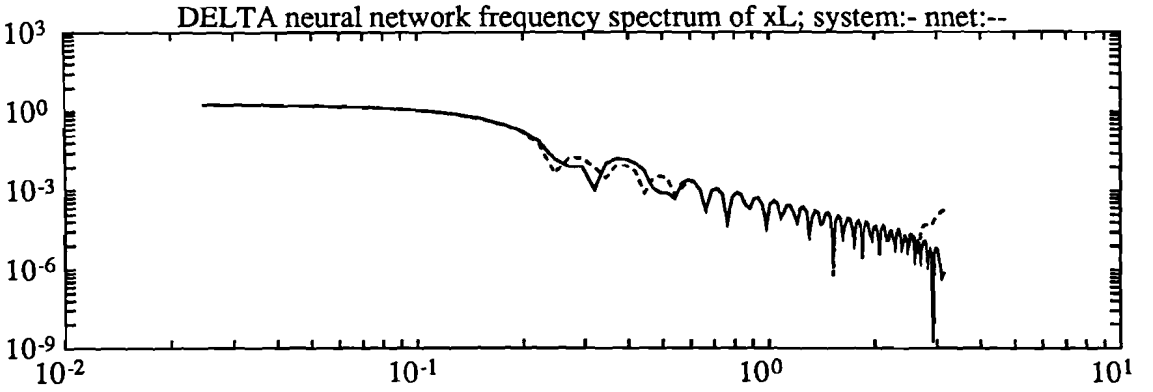
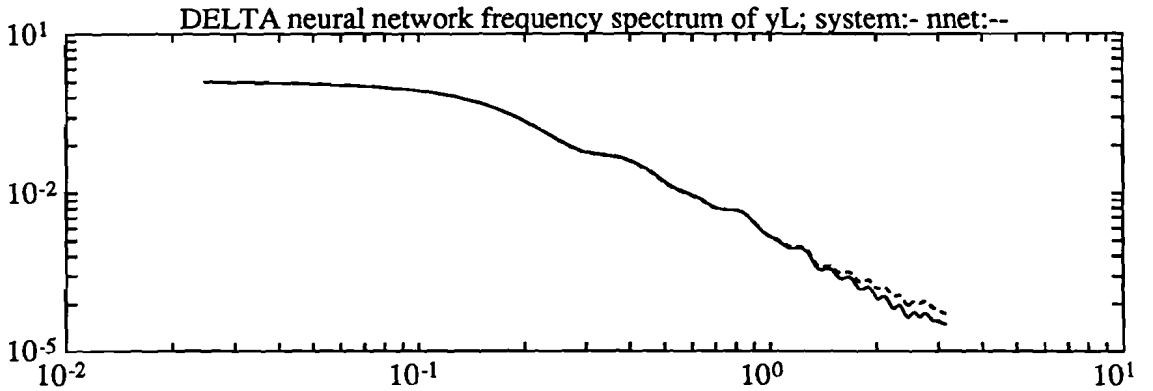


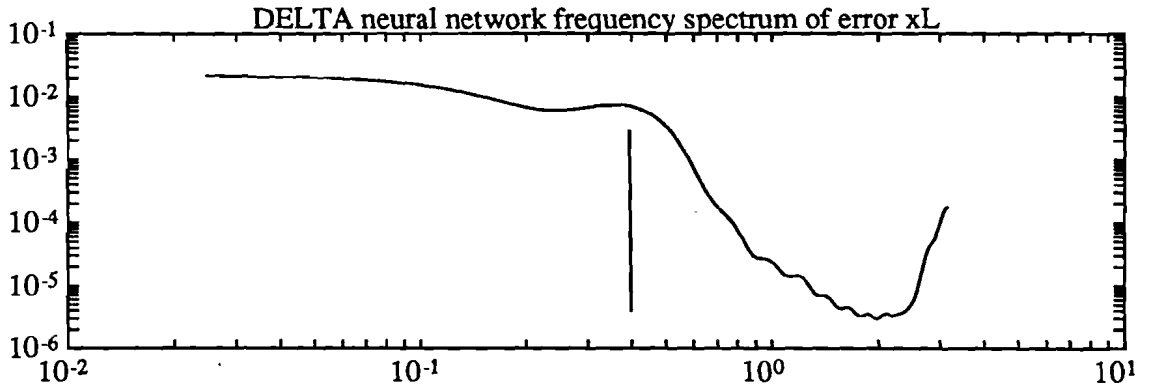
Fig.4.3 MIMO Simulation Experiment Delta



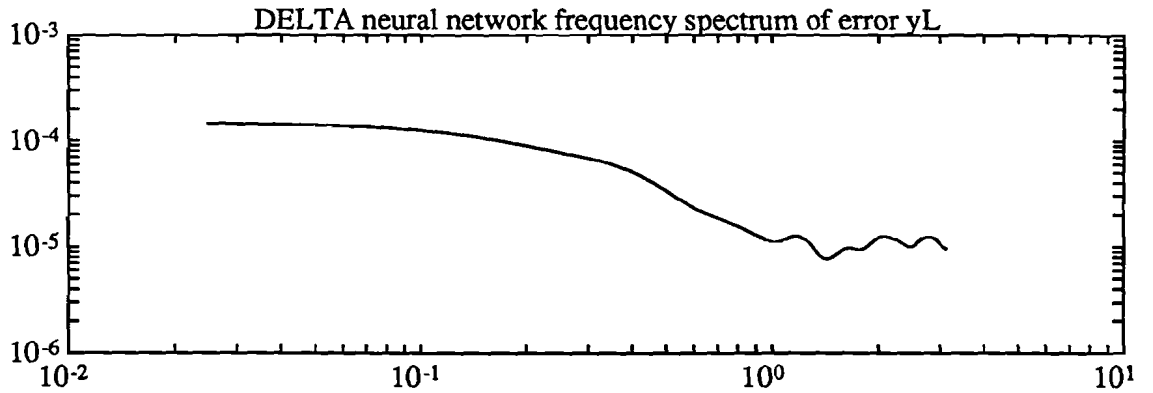
e



f



g



h

Fig. 4.3 MIMO Simulation Experiment Delta (cont.)

SISO Simulation experiments

The MIMO linear system is completely decoupled, so the coupling in the nonlinear system is expected to be very small. As indicated in figure 4.4 the MIMO network can be split up into two SISO networks that can be trained separately. These can be combined to the MIMO network by taking small random initials on the interconnecting weights. Now we can put more attention to training the SISO system x_r to x_L. The performance of the MIMO neural network is exactly the summation of the performances of the SISO networks. Next the training is continued.

In our experiment the MIMO neural network $NN_{16\,8\,8\,2}$ with 226 weights is split up into two SISO networks $NN_{8\,4\,4\,1}$ with 61 weights each. The convergence rates are increased drastically.

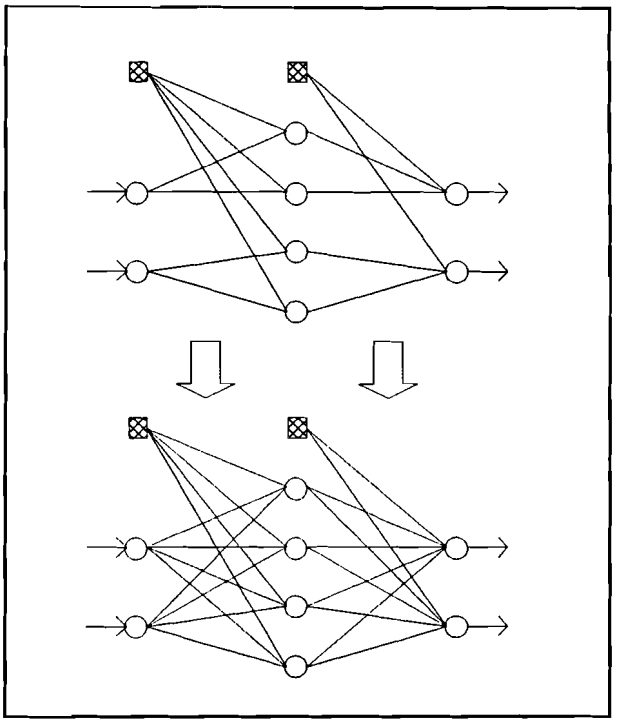


Fig.4.4. Coupling two SISO Neural Networks

Experiment

For the SISO system y_r to y_L four neural network training experiments are carried out using 1 hidden layer with 10 nodes resp. 2 hidden layers with 5 nodes each, and two resp. four delayed outputs fed back.

Starting with random initials for the weights (with the same distribution) 1000 iterations of stochastic search are followed by 100 iterations of steepest descent and 1000 iterations of Quasi Newton. For all experiments the error surface is very flat after the training experiment. The performances that were attained are:

$NN_{4\,10\,1}$	(2 delayed outputs; 1 hidden layer):	0.2%
$NN_{4\,5\,5\,1}$	(2 delayed outputs; 2 hidden layers):	0.1%
$NN_{6\,10\,1}$	(4 delayed outputs; 1 hidden layer):	4.1%
$NN_{6\,5\,5\,1}$	(4 delayed outputs; 2 hidden layers):	3.0%

With this experiment it is shown that the redundancy (only two delayed outputs are necessary) is a very important factor in making it difficult to train a neural network sufficiently. With a neural network with two hidden layers (in this experiment) a better approximation of the system is found.

Low frequency experiments

A dataset of 1000 samples for the SISO system x_r to x_L is produced with the sampling frequency of 1 Hz. The neural net NN_{8441} could easily be trained up to a performance of 0.2%. The validation and impulse response are plotted in figure 4.5. With this network the high frequency component (the swinging of the load) is not represented, so the performance is not to be compared with previous performances. Two experiments are performed making use of the knowledge that a good performance can be found using a lower sampling frequency.

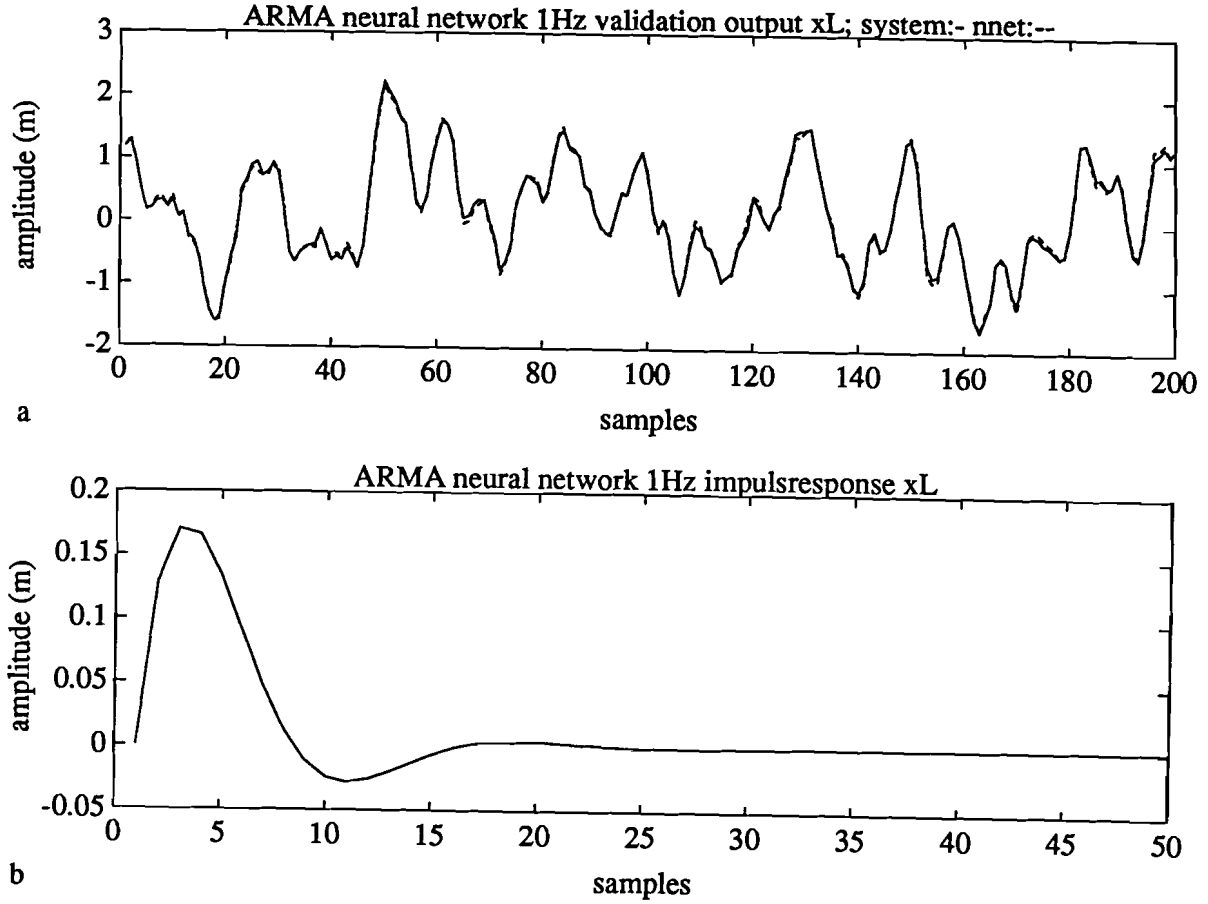


Fig.4.5 SISO Simulation Experiment x_L; 1Hz

In the first experiment (for the SISO system x_r to x_L)using 10 Hz sampling frequency we take as delayed outputs fed back to the network in stead of: $x_L(k),x_L(k-1),x_L(k-2),x_L(k-3)$, the outputs $x_L(k),x_L(k-1)$, for the high frequency component and $x_L(k-10),x_L(k-20)$ for the low frequency component.
The weights of this neural network are taken as initials for the network with other outputs fed back for the low frequency term. In stead of $x_L(k-10),x_L(k-20)$ the outputs $x_L(k-9),x_L(k-18)$. By repeating this procedure for smaller delays ($x_L(k-8),x_L(k-16)$) we could finally find a sufficient minimum for the original network with four delayed outputs.

Already in the first step no neural network representing both frequencies is found. So this experiment is not continued.

The second method is to start with the neural network model of the 1 Hz sampling frequency experiment and use the weights as initials for a 2 Hz experiment. In small steps we would proceed to the 10 Hz experiment. The results show however that the higher the sampling frequency the more difficult the learning of the network. Still the pendulum frequency is not represented.

Linear Simulation Experiments

The SISO system (x_r to x_L) can be identified in output error configuration by the MATLAB function OE that trains a linear model. This is carried out with a fourth order model. From the impulse response fig.4.6 we see that the high frequency component is represented by the model. The performance is 2% which is worse than the performance of the ARMA neural network for this output. So it is shown that the information about the higher frequency is contained in the dataset.

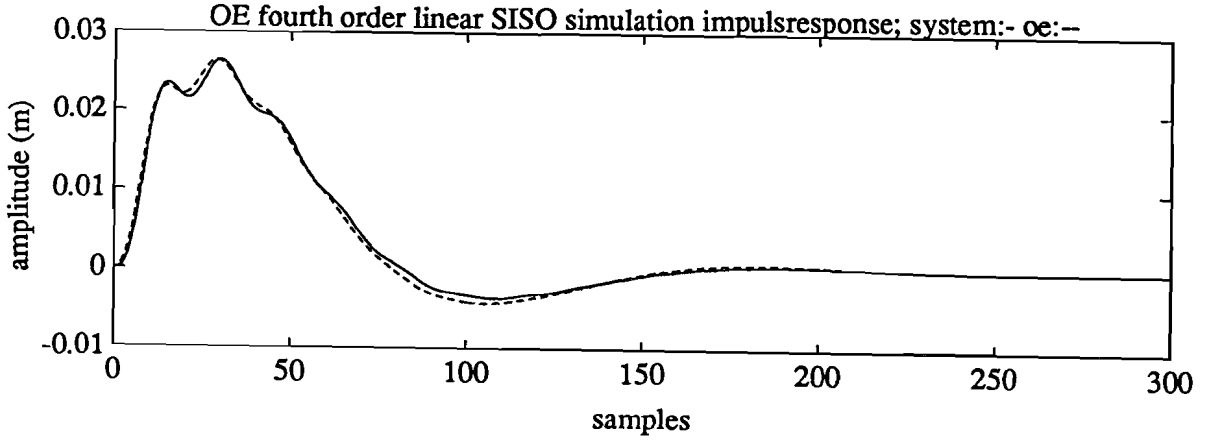


Fig.4.6 SISO Simulation Experiment x_L; Linear

4.3 Filtering to put more Weight on a Particular Frequency Range

For the SISO system x_r to x_L still the pendulum frequency component is not represented by a neural network. We will just have a look at the simulation of this SISO system. In the experiments so far it became clear that not enough weight was put on the high frequency component compared to the low frequency one, even if the performance is better than one percent. The problem is that for a good controller the angle of the load has to be identified in order to be able to suppress the swinging.

We will make sure that more weight is put on the frequency belonging to the swinging of the load during the minimization procedure. Three experiments were proposed using low pass or bandpass filters (the band around the frequency of interest).

Filtering the Input Signal

The white noise input signal is filtered before it is applied to the system. In this way the system is more excited with this particular frequency and therefore there is more power in this frequency range in the output, so more weight is put on it in the learning procedure. The neural net is trained with the filtered white noise as input.

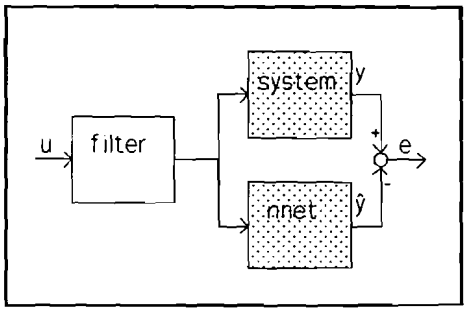


Fig.4.7 Filtering the Input

A data set is created with as input u a uniformly distributed white noise filtered with a second order bandpass Butterworth filter; this is created with the matlab function 'butter'. The frequency band is: [.6 1.2]Hz. The decay outset the band is 20 dB/decade. No better performance than 36% for the neural network was found. The validation is done with another data set with a filtered white noise input signal (with the same filter). The output of the network is plotted in fig. 4.8. Apparently only the bandfrequency was approximated.

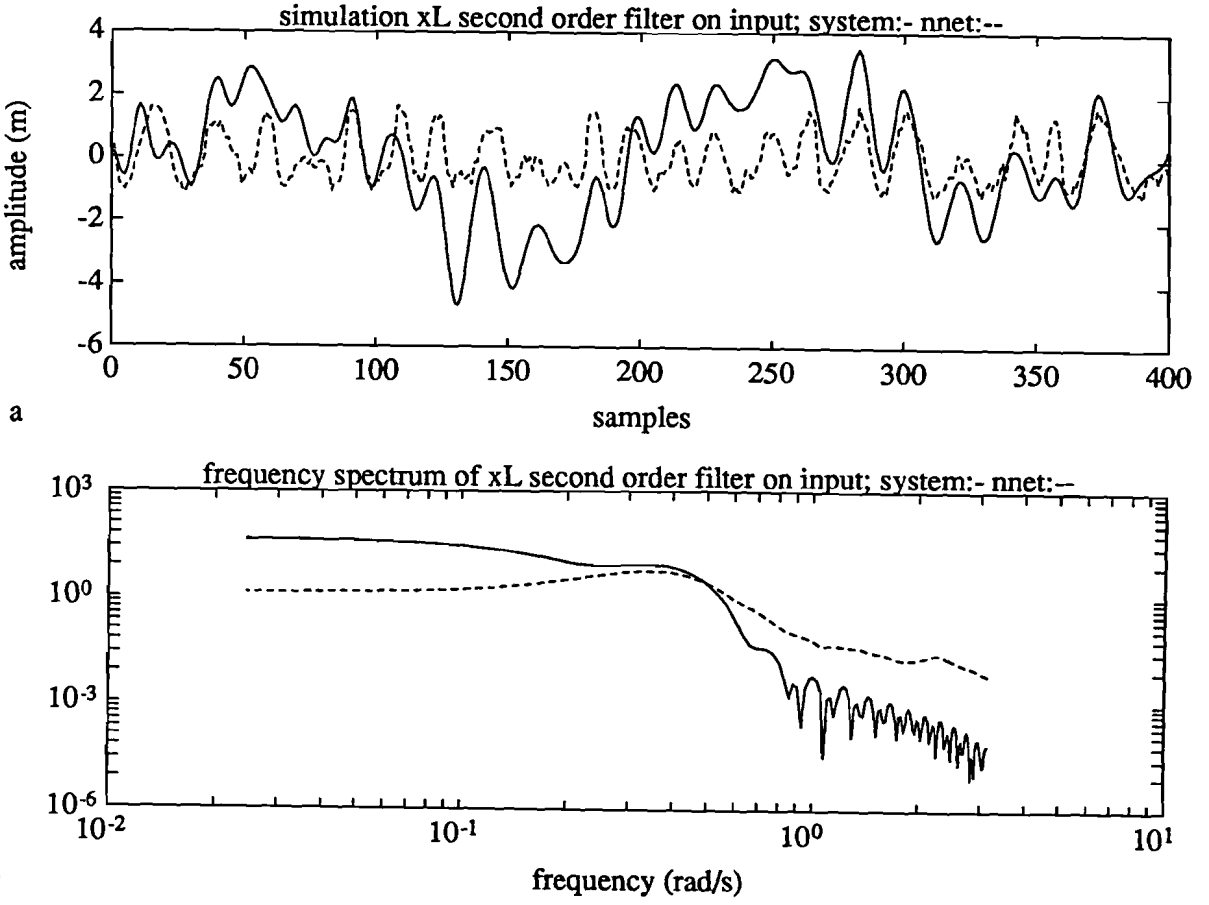


Fig.4.8 SISO Simulation Experiment; Filtering the Input Signal

Filtering the Output Signal

As indicated in figure 4.9 the outputs of the system are filtered. Now the neural net has to approximate the system including the filter. The poles of the filter are added so a higher order system has to be simulated. The neural network can be applied by filtering its outputs with the inverse filter. A problem that arises here is that the inverse filter might have poles at minus one when the filter is chosen to have zeros at minus one (low pass). Now the model might become unstable.

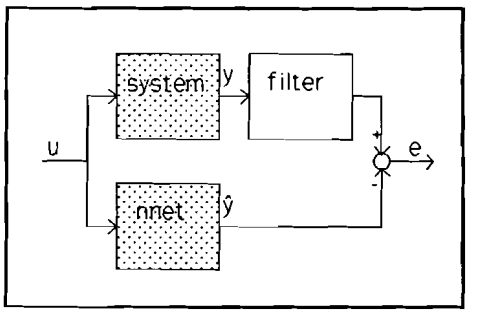


Fig.4.9 Filtering the Output

We try this method with a second and with a fourth order (decay 40 dB/decade) bandpass Butterworth filter with the same band as in the previous experiment. The neural net is $NN_{12\,4\,4\,1}$. The results are plotted in figure 4.11. The validations are done on a dataset with filtered outputs. Performances of the neural networks of 17% for the second order and 16% for the fourth order system are attained. This is very bad.

The next experiment is done with the neural net from the experiment with the second order filter. A data set is created which has an impulse as input signal. The minimization of the performance is continued only using this data set. After 25 iterations of Quasi Newton the performance is already 1e-6. The impulse response of this neural net contains both the high frequency and the low frequency component. The validation of this network on a data set with a white noise input is more than 50%. What is shown with this experiment is that it must be possible with the neural network to also approximate the high frequency component. The problem seems to be that the applied minimization routines are not accurate enough.

Filtering the Error Signal

With the configuration plotted in figure 4.10 nothing has to be changed to the data set however, the implementation of this however is very cumbersome, because back propagation has to be programmed including the filter. Another problem is that also the higher harmonics of the system would be filtered out so this method has to be combined with the original minimization routine. With the bad results of the other filtering configurations and the limited time we decided not to do this experiment.

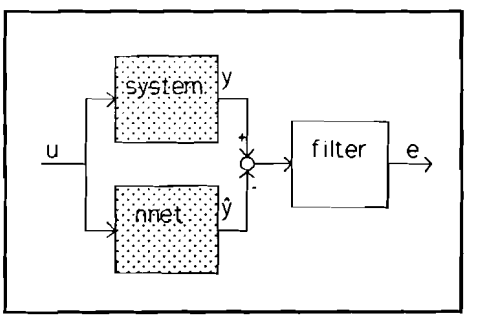
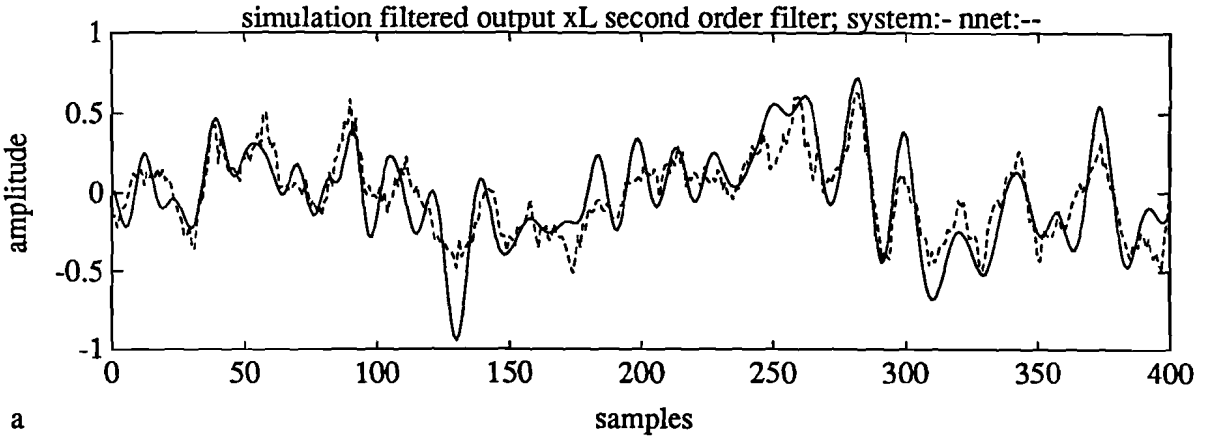
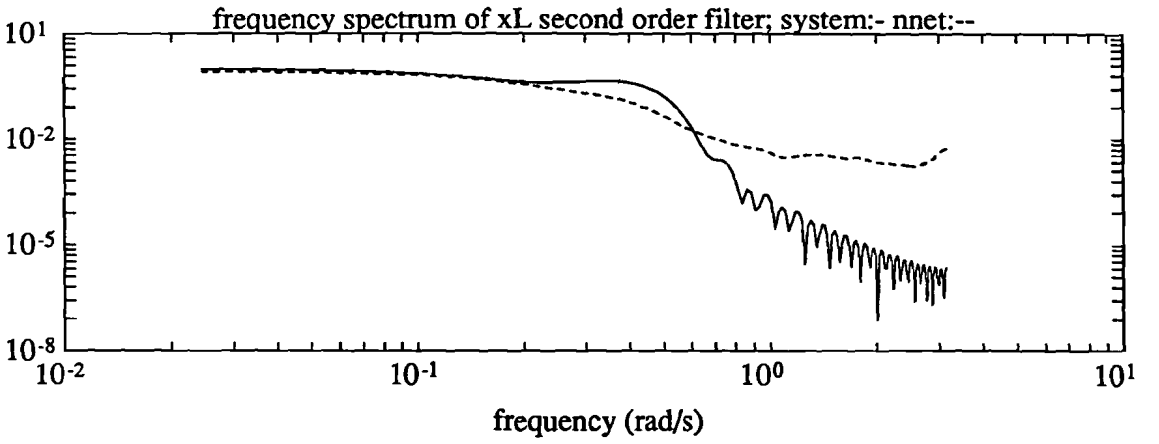
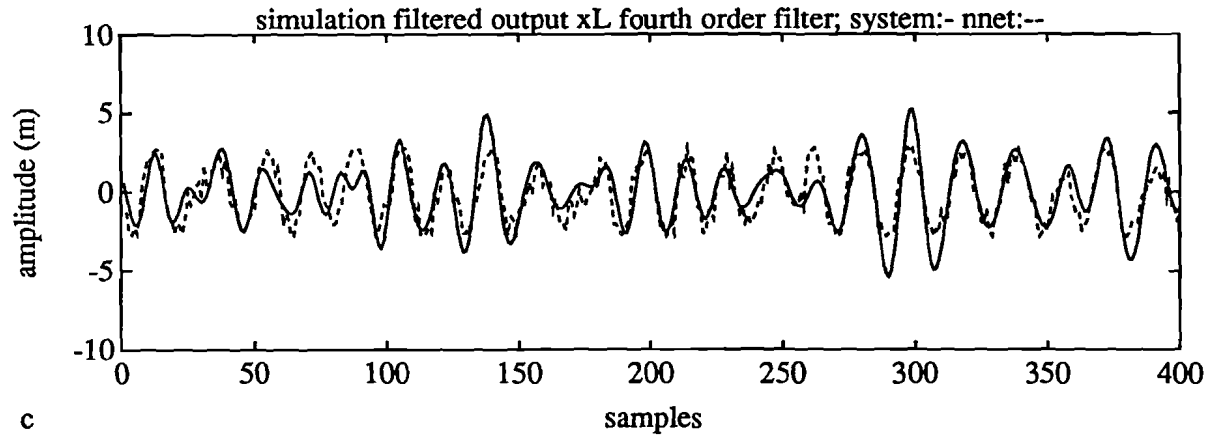


Fig.4.10 Filtering the Error







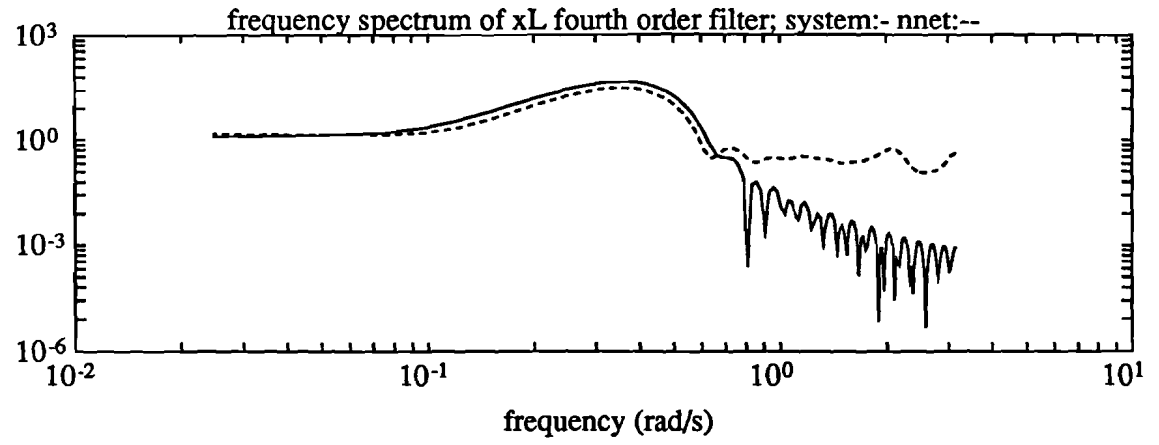


Fig.4.11 SISO Simulation Experiment; Filtering the Output Signal

4.4 Simulation Experiments of the State Space System

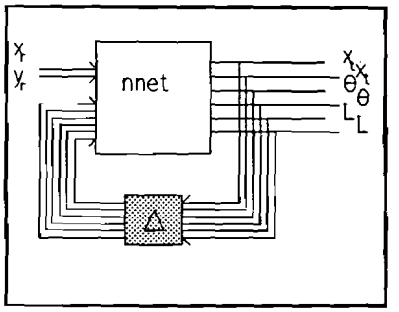


Fig.4.12 Sate Space Repr.

Still we didn't manage to simulate the overhead crane using the configuration with two inputs and two outputs. We release the wish to use just the two outputs x_L and y_L and try to simulate it using the six states as plotted in figure 4.12. The advantage is that in the states all the necessary information about the system is directly available to the neural net. Now we can put more weight on the state θ by simply scaling this output. The outputs have to be fed back with only one delay.

The neural network $NN_{8\ 10\ 10\ 6}$ is trained on a dataset of 5000 samples. θ is calculated in radians because then the amplitudes of the outputs are directly comparable. As we want to put more weight on the swinging of the load in the training of the network the state θ is scaled with a factor 5 (scaling as in(2.11) is not applied). The performance of the neural network for each of the six outputs is: x:18%; $\dot{x}$:31%; θ:15%; $\dot{\theta}$:20%; L:35%; $\dot{L}$:19%. The outputs x_L and y_L can be calculated with the states. In figure 4.13 these outputs are plotted and compared to the system outputs. The frequency spectrum of the output of the model for each of the states is plotted in figure 4.14.

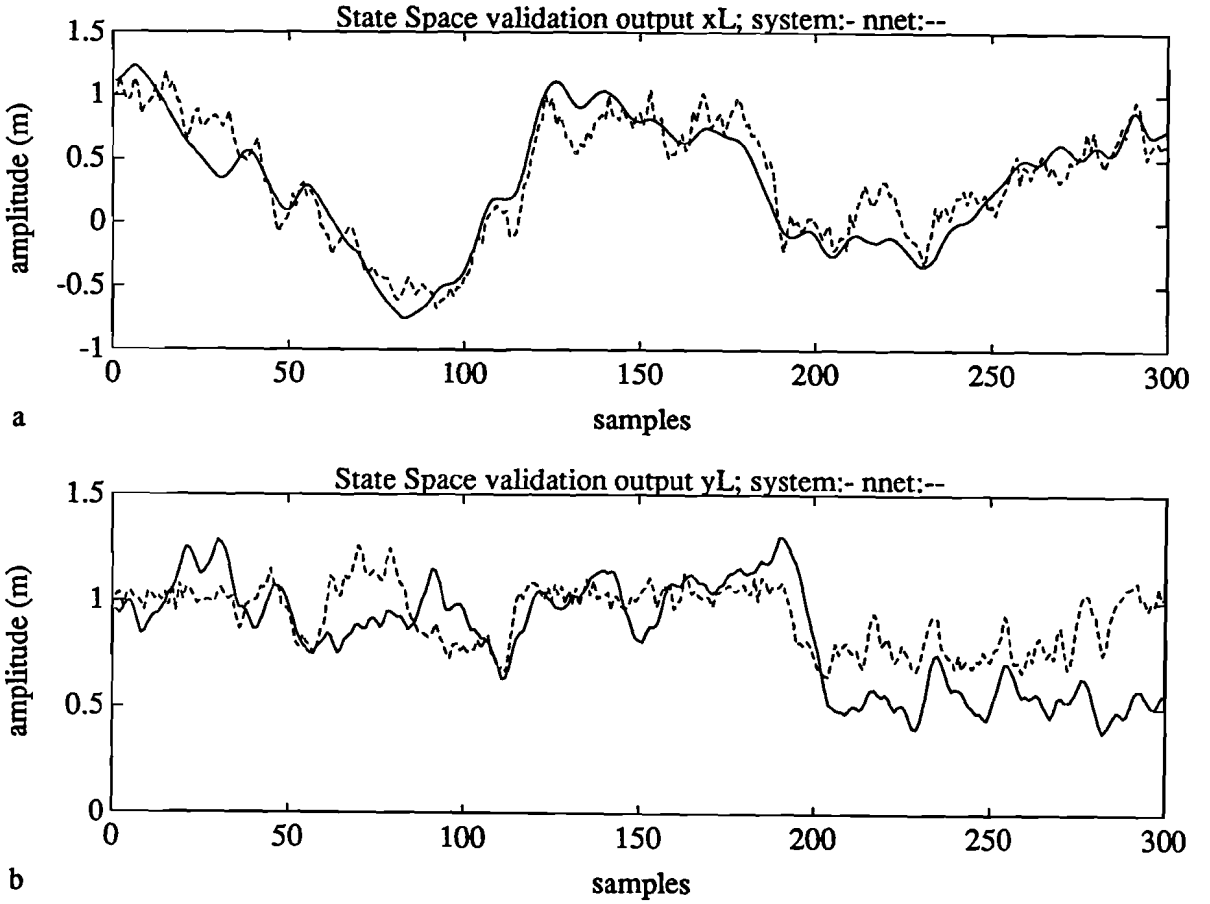


Fig.4.13 State Space Simulation Experiment; Outputs after Conversion

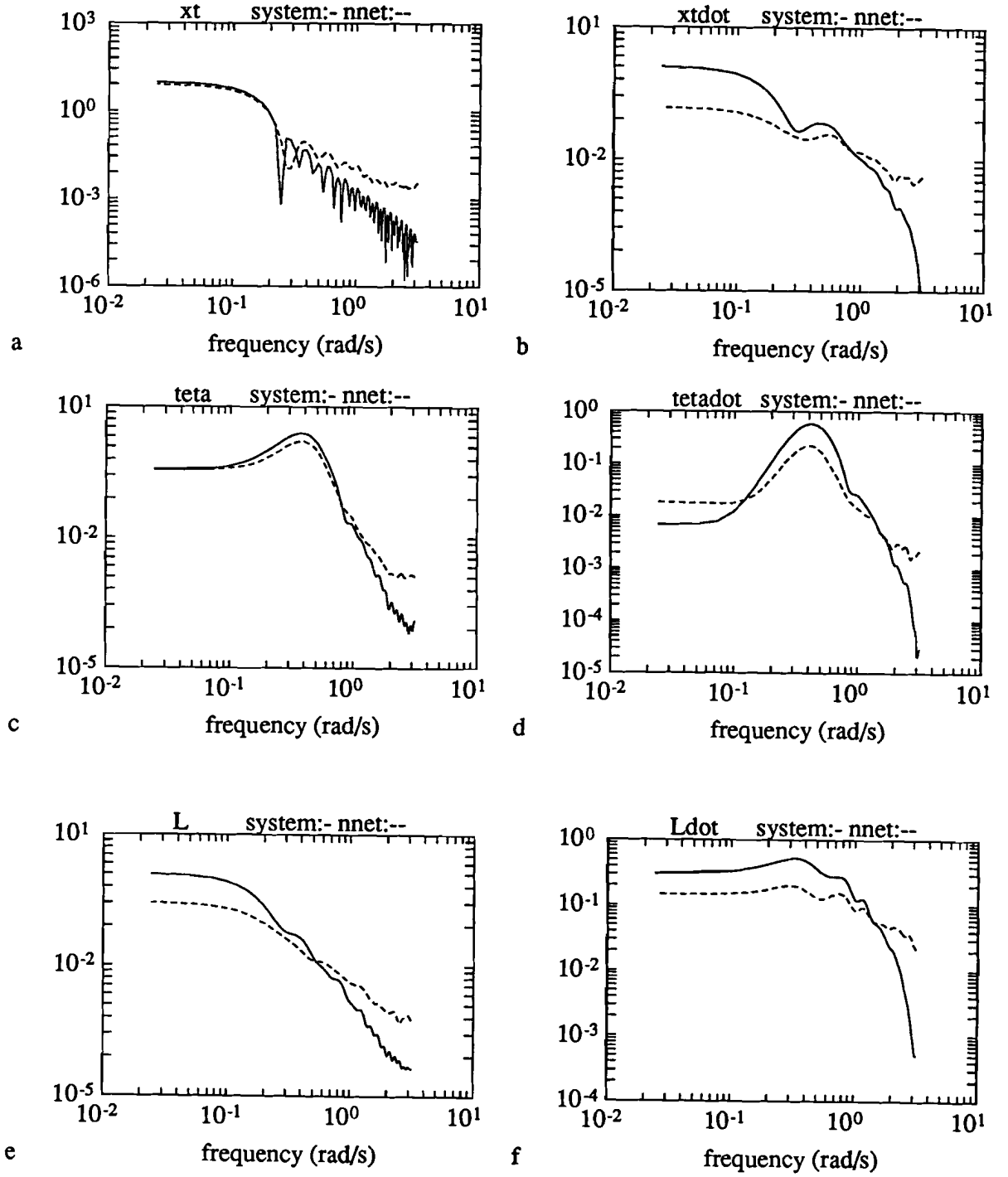


Fig.4.14 State Space Simulation Experiment; Spectra States

We would expect to find a better simulation model with the state space than with the 2-input 2-output representation. Now the overall performance of the neural network using the state space representation is worse, 25% against 1.1% for the ARMA 2-output neural network (the comparance is not completely fair because another criterion was minimized).

There are three possibilities for the bad result:

1: The dataset does not contain sufficient information about the system
2: Because of the size of the neural network it cannot represent the system
3: The training methods of the neural network are not powerful enough

The first possibility we can exclude, because more information is available than for the 2-input 2-output system. A fourth order linear simulation system even could represent both frequencies appearing in x_L.

For the second possibility we have a closer look at the number of nodes that is necessary to approximate this MIMO six output system.

If for approximation of the SISO function $y=f(x,0)$ we need n nodes (n is an integer). The question is how many nodes we need to approximate the MIMO function $f(x_1,x_2)$. If the function is completely decoupled we need 2n nodes, but this is the absolute minimum.

If we want to be sure that the neural network is able to represent the MIMO state space system exactly we need to know how many nodes there are necessary.
If the sampling frequency is high enough we can equate $x(k)-x(k-1)\simeq\dot{x}$. For calculation of the six states $x_t,\dot{x}_t,\theta,\dot{\theta},L,\dot{L}$ all the necessary information is included in $x_r,y_r,x_t,\dot{x}_t,\ddot{x}_t,\theta,\dot{\theta},\ddot{\theta},L,\dot{L},\ddot{L}$. From the nonlinear system equations (2.1) we know that $\ddot{x}_t,\ddot{L},\ddot{\theta}$ can be calculated, besides some linear functions, with the nonlinear functions $y_r\sin(\theta)$, $\sin(\theta)/L$, $\dot{x}_t\cos(\theta)/L$, $\dot{L}\dot{\theta}/L$, $\cos(\theta)$, $L\dot{\theta}^2$, $\ddot{x}_t\sin(\theta)$ (For convenience we neglected the linear feedback that prestabilizes the system). Multiplications and divisions will have to be done by the network.

For simulation of a multiplication the network needs at least two layers. This is because the output of a multiplication has a saddle point, it is positive in the first and third quadrant, negative in the second and fourth quadrant so we have the same problem as in the experiment in section 3.4. With one layer only one straight line boundary between a positive and a negative region can be determined. The divisions can be approximated with a network with one layer because no saddle points are generated.

The most difficult term to be approximated is $\ddot{x}_t\cos(\theta)/L$ for which $\ddot{x}_t$ is needed. For $\ddot{x}_t$ first $y_r\sin(\theta)$ is needed; see the equations (2.1). So the network needs one layer to generate $\sin(\theta)$, followed by two layers for the multiplication with y_r, two layers to multiply $\dot{x}_t\cos(\theta)$ and one layer for division by L which makes 6 layers in total. Also each layer needs a minimum number of nodes to transfer all the information through the

network. When each layer has 10 nodes this leads to a gigantic network $NN_{8\ 10\ 10\ 10\ 10\ 10\ 6}$ with 704 weights. This is so big that it can hardly be trained with the training algorithms that are used now.

On the other hand it would be easier to equate $\ddot{x}=\dot{x}(k)-\dot{x}(k-1)$. If this difference is made in the first layer the network would need two layers extra to make $\ddot{x}_t\cos(\theta)/L$ (arbitrary functions can be approximated). Now three layers are enough for the network, but in each layer more difficult functions must be approximated so more nodes are necessary. When we expect 16 nodes per hidden layer this leads to the network $NN_{8\ 16\ 16\ 16\ 6}$ with 790 weights.

So when the number of layers is decreased the number of nodes per layer is increased and the number of weights of the network doesn't decrease.

To show that a better neural network approximation must be possible with the network $NN_{8\ 10\ 10\ 6}$ an experiment is set up with linear processing functions. With the neural network with one hidden layer and six nodes $NN_{8\ 6\ 2}$ the performance for each output is: x:13%; $\dot{x}$:11%; θ:14%; $\dot{\theta}$:19%; L:0.1%; $\dot{L}$:14%. With this experiment it is shown that a network with one hidden layer with 6 nodes is not big enough, but already a better result is obtained than with the network using sigmoids with 10 nodes; the performance is 8% in stead of 25%, so certainly a better approximation must be possible with that network.

4.5 Extra Experiment Using ARMA Representation Neural Nets

The approach mentioned in chapter 3, formula (3.13) is tested on an ARMA representation neural net $NN_{18\ 5\ 5\ 2}$. The same dataset is used as in the experiment in section 4.1. The method proves to be successful as can be seen from the improvement of the performance: Here 0.2% (simulation; for the validation it is 0.8%) against 1.1% (validation) in the original ARMA experiment. The problem remains that the high frequency component is not represented, as can be seen in the impulse response plotted in figure 4.15.b.

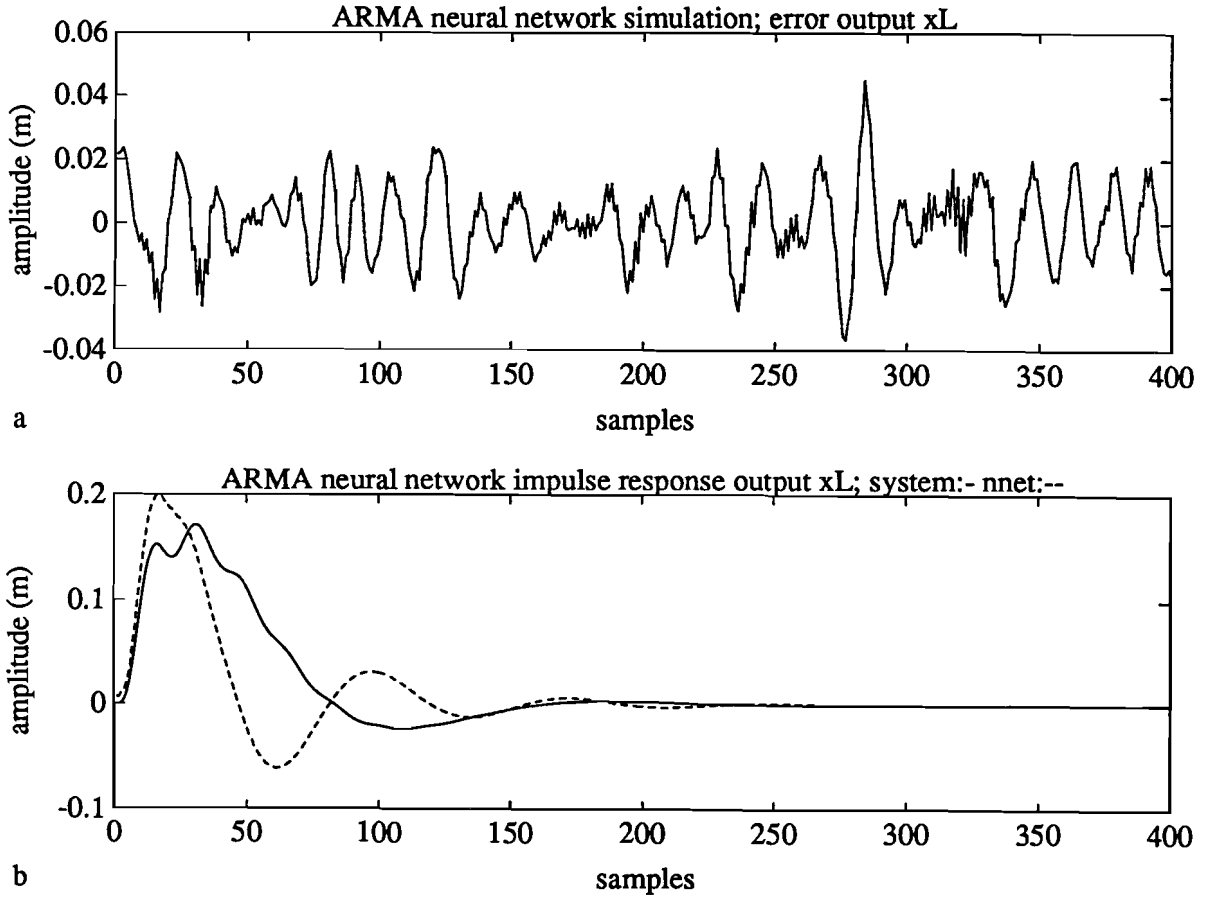


Fig.4.15 Extra MIMO Simulation Experiment ARMA

CHAPTER 5
Conclusions and Recommendations

Conclusions

The MIMO overhead crane is a complex rather nonlinear system. A simulation model of it is created. The model is not validated because the real crane is not yet available.

It is attempted to simulate the system with a neural network model. After the training of the network the output error performance of the model is bad and the model doesn't represent the swinging of the load. With this model no good controller can be built because the error is too big and the very swinging of the load that is not represented has to be suppressed by the controller.

The applied neural network gradient and stochastic training algorithms are not powerful enough to find a network representing all the necessary information of this complex nonlinear MIMO system. With different approaches we tried to speed up and facilitate the training routines.

The training of the network is scarcely improved when the delta representation method is applied. By using filtering of input- or output signals or by using the state space representation of the system the training is more problematic.

To guarantee that the neural network can accurately approximate the system a very big network is necessary. The size of the network has to be kept as small as possible to increase the rate of convergence, because a smaller network has less redundancy. Doing this the network is made smaller than allowed.

The problem is that we have to choose a size of the network that is not too small to make sure that the network can represent the system up to a desired degree of accuracy. On the other hand the size of the network has to be kept small, to increase the convergence rate and the probability of success of the training algorithms. The minimization algorithms are not powerful enough to guarantee that a complex nonlinear MIMO system can be simulated sufficiently accurate with a neural network model.

Recommendations

With the applied algorithms we are not able to train a neural network that represents a complex MIMO non linear system. In many directions study will have to be done.

Make sure that the neural network is able to represent the system. Find a guarantee that the neural network has enough layers and nodes per hidden layer to represent the system. Maybe this can be found analytically by splitting up the complex non linear system into basic functions for which a minimum neural network size (to represent it) can be defined.

Improve the training methods of the neural network. Both the rate of convergence and the probability of success to find a proper neural network model must be increased. Use can be made of the fact that the network can be composed of smaller neural networks (that can be trained separately) for which the convergence rate is larger and that linear functions can be better minimized with the optimization methods.

To create a simulation model more information is available about the system than applied here. Try to make use of the extra information.

When a neural network model is obtained still a neural network controller will have to be trained. It is also possible to use a neural network controller that does not make use of a neural network simulation model. It could be trained on the Simulink simulation model of the crane.

REFERENCES

[1] Beemt van den, B.J.M.
Nonlinear System Identification of a Gantry Crane Process Using Neural Networks
Department of Electrical Engineering, Measurement and Control Section, Eindhoven University of Technology, december 1992
M.Sc. Thesis

[2] Chen, S. et al.
Nonlinear System Identification Using Neural Networks
Int.J.Control, Vol.51(1990), no.6, pp.1191-14

[3] Hunt, K.J. et al.
Neural Networks for Control Systems - A Survey
Automatica, Vol.28(1992), no.6, pp.1083-12

[4] Hunt, K.J., Sbarbado, D.
Neural Networks for Nonlinear Internal Model Control
IEE Proceedings-D, Vol.138(1991), no.5, pp.431-38

[5] Jacobs, R.A.
Increased Rates of Convergence Through Learning Rate Adaptation
Neural Networks, Vol.1(1988), no.1, pp.295-07

[6] Minsky, M., Papert,S.
Perceptrons: An Introduction to Computational Geometry
MIT Press, 1969

[7] Moonen, F.J.
Neural Nets in an Acoustic Application
Department of Electrical Engineering, Measurement and Control Section, Eindhoven University of Technology, 1992
M.Sc. Thesis

[8] Moonen, F.J.
Software Manual for MIMO Identification Using Artificial Neural Nets
Department of Electrical Engineering, Measurement and Control Section, Eindhoven University of Technology, 1993

[9] Morari, M., Zafiriou, E.
Robust Process Control
Prentice-Hall, 1989

[10] Narendra, K.S., Parthasarathy, K.
Identification and Control of Dynamical Systems Using Neural Networks
IEEE Trans. on Neural Networks, Vol.1(1990), no.1, pp.4-27

[11] Narendra, K.S., Parthasarathy, K.
Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks
IEEE Trans. on Neural Networks, Vol.2(1991), no.2, pp.252-62

[12] Narendra, K.S., Mukhopadhyay, S.
Intelligent Control Using Neural Networks
IEEE Control Systems, vol.12(1992), pp.11-18

[13] Pijnenburg, J.L.C.M.
Nonlinear System Identification Using Neural Networks
Department of Electrical Engineering, Measurement and Control Section, Eindhoven University of Technology, 1992
M.Sc. Thesis

[14] Qiu, M. et al.
Accelerated Training of Backpropagation Networks by using Adaptive Momentum Step
Electronics Letters, Vol.28(1992), no.4, pp.377-79

[15] Schreppers, M.A.M.
Control of a Gantry Crane Process by a Fuzzy Logic Controller
Department of Electrical Engineering, Measurement and Control Section, Eindhoven University of Technology, 1993
M.Sc. Thesis

[16] Telkamp, H.J.M., Damen, A.A.H.
Neural Network Learning in Nonlinear System Identification and Controller Design
European Control Conference 1993, Groningen, pp.798-04

[17] Willis, M.J. et al.
Artificial Neural Networks in Process Estimation and Control
Automatica, Vol.28(1992), no.6, pp.1181-87

[18] Youji, I. et al.
A Nonlinear Regulator Design in the Presence of System Uncertainties Using Multilayered Neural Networks
IEEE Trans. on Neural Networks, vol.2(1991), no.4, pp.410-17

LIST OF SYMBOLS

a_r	radial acceleration
a_ϕ	tangential acceleration
d_L	L-directional damping constant
d_x	x-directional damping constant
d_θ	θ-directional damping constant
f_{pend}	pendulum frequency
g	constant of gravity
F_h	hoisting force
F_t	x-directional force on the trolley
L	length of the cable
L_0	length of the cable in the linearization point
m_L	mass of the load
m_t	mass of the trolley
x_L	x-position of the load
x_r	reference input x position of the load
x_t	x-position of the trolley
y_L	y-position of the load
y_r	reference input y position of the load
A,B,C,D	state space matrices
Q,R	LQR matrices
K_{LQR}	matrix containing LQR feedback variables
K_{LOOSE}	matrix containing loose feedback variables
$NN_{i_0 i_1 \ldots i_L}$	neural network with i_0 inputs and i_L outputs
N	number of samples
L	number of layers
J	performance of the neural network
J_{LQR}	LQR performance
y	system output
$\hat{y}$	neural network output
u	system input
e	error signal
r	reference signal
w	vector of weights
w_j	j^{th} weight
p	number of outputs
ndo	number of outputs fed back
H	Hessian
$\nabla_w J(w)$	gradient of the performance to the weights
α	learnvelocity factor
σ or z^{-1}	shift operator