TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Dynamic neural adaptive control for (partial) unknown linear and nonlinear systems using feedback-error-learning

Citation for published version (APA):
Verdijck, G. J. C. (1995). *Dynamic neural adaptive control for (partial) unknown linear and nonlinear systems using feedback-error-learning*. (DCT rapporten; Vol. 1995.127). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1995

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Nov. 2023

Dynamic Neural Adaptive Control for (Partial) Unknown Linear and Nonlinear Systems using Feedback-Error-Learning

G.J.C. Verdijck

Report number: WFW 95.127

Practical Assignment Report

Author:	G.J.C. Verdijck
Institution:	Eindhoven University of Technology
	Department of Mechanical Engineering
	Fundamentals of Mechanical Engineering (WFW)
Mentor:	dr. ir. M. J. G. v.d. Molengraft
Report number:	WFW 95.127
Date:	August 1995

Dynamic Neural Adaptive Control for (Partially) Unknown Linear and Nonlinear Systems using Feedback-Error-Learning

G.J.C. Verdijck

Mechanical Engineering, Eindhoven University of Technology, The Netherlands

ABSTRACT

In this paper a dynamic neural adaptive controller using *Feedback-Error-Learning* will be proposed which is capable of controlling partially unknown or totally unknown systems. The controller consists of a neural controller and a conventional feedback controller. The neural controller functions as the adaptive feedforward part and the conventional controller as the feedback part. The feedback controller is essential for the feedback-error-learning. Contrary to most other control schemes it is not necessary to go through an identification process first to learn the dynamics of the unknown system before using the controller. The basic element in the neural adaptive scheme is the so-called *dynamic neural unit* (DNU). The learning of the neural unit develops during the control of the (partially) unknown system. The neural network gradually takes over from the conventional controller with fixed-gains. The effectiveness of this adaptive control scheme is demonstrated by both a computer simulation study and experiments.

1. Introduction

Mostly when the system to be controlled is unknown or partially unknown, a mathematical model of the system is derived using system identification techniques, i.e., knowledge on the system to be controlled is obtained and the parameters for the neural controller are determined.

When the goal is to control the system and not to obtain a correct model of the system to be controlled, it is possible to perform the control in a different way. A plant inverse is identified and used in control as can be seen in figure 1.

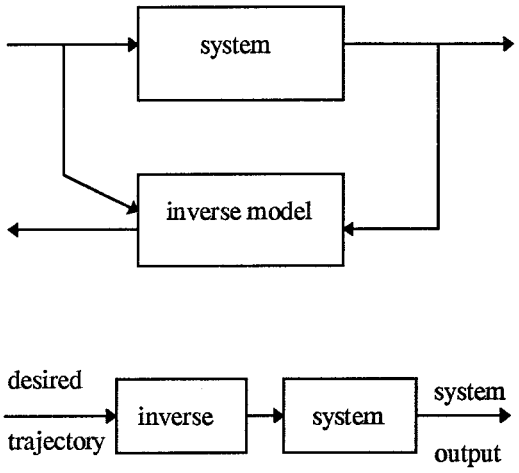


Figure 1 : identifying a system inverse

Identifying the inverse, the input of the network is the plant output and the plant input serves as target output of the network. When the network can be trained to match these targets, the plant inverse is identified. This inverse can be used for control purposes. Kawato (ref. [2]) proposed a method called feedback-error-learning. With this method the inverse is constructed while control takes place. Another difference is the use of the desired plant output instead of the actual output as input signal for the neural network. This means that the neural network can be seen as an adaptive feedforward part of the controller. Adaptive because the parameters of the neural network are adjusted during control. This adaptive algorithm uses an error signal which is computed by a feedback controller. A major problem with the identification of an inverse model of the plant arises when many plant inputs produce the same output, i.e., when the plant's inverse is ill-defined. In this case, the network will attempt to map the same network input to many different target responses. Nevertheless, the use of nonlinear networks for

identifying nonlinear plant inverses can be of interest because of the immediate use for control.

Optimal control methods apply if there is a model of the plant together with a performance measure that both are sufficiently accurate and tractable. In less structured situations it may be possible to use on-line learning methods (ref. [3]). Reinforcement learning addresses the problem of improving performance as evaluated by any measure whose values can be supplied to the learning algorithm. With a supervised system a performance measure is defined in terms of a set of targets by means of a known error criterion. With reinforcement learning the learning algorithm is not told what the desired control signals are that lead to optimal plant performance. Reinforcement learning tries to determine the target controller outputs, or desired changes in controller outputs, that would increase plant performance. The plant performance is evaluated by a critic function which produces a measure which is supplied to the learning algorithm as can be seen in figure 2. Reinforcement learning essentially involves two problems. Finding a critic function capable of evaluating plant performance that is informative enough to allow learning. And second, determining how to alter the controller outputs to increase the plant performance.

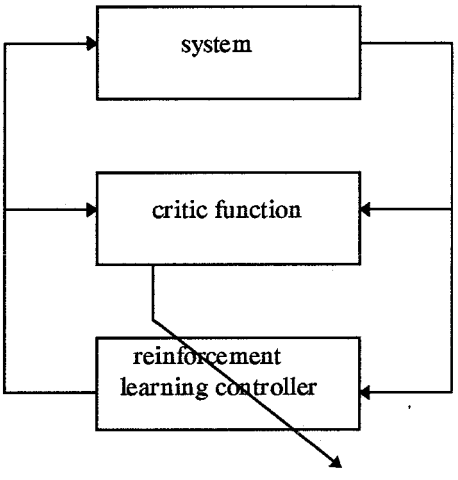


Figure 2 : a reinforcement learning control system

The idea of a critic represents an adaptive approach to optimal control by means of on-line learning a subgoal performance measure that is consistent with control objectives. If the performance measure is an error based on known targets, the desired controller outputs are known. This means that reinforcement learning can solve the same problem a supervised method would solve, but without making use of the knowledge available. As a consequence the learning time is longer than it would be with a more specialized supervised method. But when these more specialized methods are not available, reinforcement learning presents a good solution. One could say that supervised methods can always be used when a plant is unknown. An identification process delivers a model on which supervised control can be based. Because model based methods need not to be successful with an inaccurate model, reinforcement learning, which does not need a model of the plant, can be applied for control purposes.

The feedback-error-learning method can be seen as a kind of critic design, in which a preprogrammed feedback controller acts as a (nonadaptive) critic. In feedback-error-learning the feedback controller is also part of the total applied motor command (figure 3), in contrast with reinforcement learning where the adaptive critic is only an evaluation signal which does not take part in the actual control (figure 2). The feedback motor command is used as an error signal to train a neural network which generates a motor command. As is stated before, the neural network learns an inverse model which produces the desired motor commands. In figure 3 the proposed Control scheme is shown.

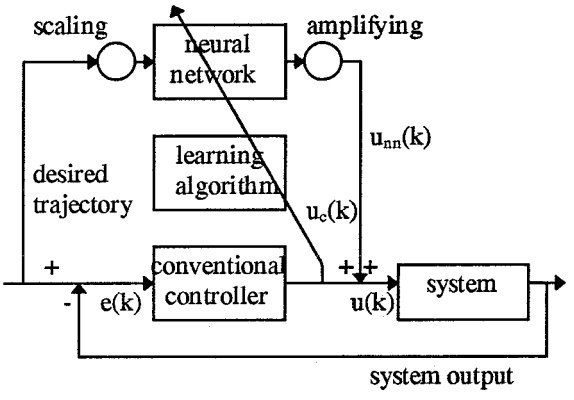


Figure 3 : the control scheme

A perfect feedforward control can be realized if the feedforward controller provides a well-defined inverse model of the controlled system. The problem is to find a motor command which realizes the desired trajectory. The feedback controller converts the error in state-space into an error in the motor command-space which is used in the learning algorithm for the neural network. The actual motor command is the sum of the feedback and the feedforward commands. This because of the trajectory stability guaranteed by the feedback

controller. When the difference between the actual motor command and the desired motor command is known, various supervised learning rules can be used to train the neural network. However, since the dynamics of the controlled system are not known, this desired motor command and the trajectory error are both unknown. This is the reason why the supervised learning rules can not be used and another learning algorithm must be considered. In Section 3 the learning algorithm of the proposed control scheme is defined.

As neural network, a little network consisting of just one DNU (ref. [1] & [4]) will be considered. It would be possible to use a neural network consisting of more layers and/or using more input units. This can be a subject of future research. Because of the chosen activation function a possible problem could be that the neural output lies between -1 and +1. Therefore scaling of the desired trajectory and amplifying the neural output is necessary.

In Section 2 the architecture of the DNU will be explained, followed by the derivation of the learning and adaptive algorithm in Section 3. In Section 4, the conventional controller will be briefly discussed. After these elements are described, in Section 5 some remarks regarding stability and convergence are made. Section 6 describes the control scheme in detail including the amplifying term. In Section 7 the computer simulation results for this control scheme are shown and discussed. Two models are considered, a *simple system* without non-linearity's and a *two mass system* with friction. This system is besides being used in simulations an object of experiments. These results are discussed in Section 8. At the end of this article some conclusions follow on what can be achieved with this control scheme and what aspects have to be investigated in future.

2. The architecture of the DNU

The neural unit, the DNU (ref. [1] & [4]), consists of two delay elements with feedforward and feedback weights, and a time-varying nonlinear activation function. The dynamics of the neural model can be described by the following difference equation

$$v_1(k) = -b_1 v_1(k-1) - b_2 v_1(k-2) + a_0 x(k) + a_1 x(k-1) + a_2 x(k-2) \quad (2.1)$$

where the neural input x(k) represents the desired state, and v_1(k) is the neural state.

The neural unit used in this control scheme is a so-called recurrent unit. In this case it means that feedback signals are used within the unit itself. Therefor a neural state v_1 consists. The feedback is necessary to have memory. Without memory the output is solely determined by the current inputs and the values of their weights. Besides these synaptic weights there is the weight g_s, the somatic gain. This somatic gain determines the slope of the activation function (2.2). The neural output can be described as

$$u(k) = \tanh(v(k)) \quad (2.2)$$

in which v(k) can be described as

$$v(k) = g_s v_1(k) \quad (2.3)$$

The weights and the gain are the adjustable parameters of the DNU and these are updated every iteration. On account of the definition (2.2), the neural output u(k) lies between -1 and +1. The concept of varying the gain g_s during the process of learning and adaptation is called somatic learning. The algorithms to modify these somatic and synaptic weights are derived in the following section. Figure 4 shows the architecture of the DNU.

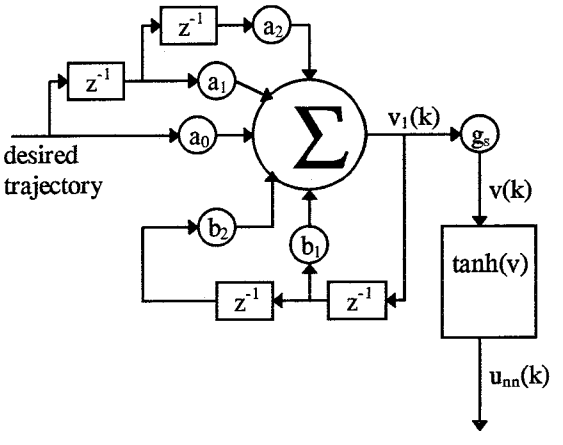


Figure 4 : the architecture of the DNU

3. Learning and adaptive algorithm

The goal of the learning and adaptive algorithm is to optimize the parameters to maximize the performance of the controller. The parameters are modified in each iteration to cause the system output to approach the desired state. The trajectory error e(k) is defined by

$$e(k) = x(k) - y(k) \tag{3.1}$$

in which y(k) is the system output. This trajectory error e(k) must go to zero to cause the system output to approach the desired state.

The problem is to calculate the desired motor command so that the desired trajectory is realized. The desired motor command can be described by

$$u_{nnd}(k) = F(\ddot{x}(k), \dot{x}(k), x(k)) \tag{3.2}$$

Because the dynamics of the controlled system are (partially) unknown, the desired force as defined in (3.2) is also unknown. The actual motor command, which is the system input u(k), is defined by

$$u(k) = u_c + u_{nn}(k) \tag{3.3}$$

where the neural output can be described as

$$u_{nn}(k) = \psi(p, x(k)) \tag{3.4}$$

where p are the various weights of the neural unit. The neural output depends on the values of the various parameters and the desired trajectory as explained in Section 2.

ASSUMPTION I: the inverse dynamics of the controlled system can be described by the neural network given a suitable set of parameters.

A performance measure is defined which has to be optimized with respect to the various parameters. To force the system output y(k) to follow the desired trajectory x(k), the force $u_{nnd}(k)$ is necessary which is the desired motor command. The aim is to learn the network to produce this desired command and after learning the feedback force will be zero. A performance measure is defined by

$$J = \frac{1}{2} u_{imag}{}^{T}(k) u_{imag}(k) \tag{3.5}$$

in which $u_{imag}(k)$ is defined by

$$u_{imag}(k) = u_{nnd}(k) - u_{nn}(k) \tag{3.6}$$

This defines the error in the neural output. By minimizing this motor command error, the network motor command u_{nn} approaches the desired motor command u_{nnd} and the feedback motor command decreases. When the applied force u(k) equals the needed force $u_{nnd}(k)$, the state-space error (3.1) will be minimized.

It is possible to use a different performance measure (3.14). The aim of this performance measure is to minimize the feedback force and the plant output following the desired trajectory. But when this measure is used in the algorithm to modify the parameters there is a problem in controlling (partial) unknown plants. Differentiating the applied force u(k) with respect to the various parameters is the problem. This is often solved by simply set this term to zero. Because the performance measure defined by (3.5) is more elegant, it is used to define the adaptive algorithm.

The feedback motor command $u_c(k)$ approximates the error in the motor command-space and can be seen as a kind of adaptive critic which reflects the quality of the neural network. In common adaptive critics a term must be maximized and the adaptive critic does not directly take part in control, contrary to the method of feedback-error-learning. After learning, the feedback motor command will be small and the DNU acts as the main controller.

The slope of the performance measure with respect to the various parameters is needed to optimize the parameters based on the steepest-descent algorithm

$$\frac{\partial J}{\partial p} = \partial\left(\frac{1}{2} u_{imag}{}^{T}(k) u_{imag}(k)\right) / \partial p \tag{3.7}$$

$$\frac{\partial J}{\partial p} = u_{imag}(k) \frac{\partial u_{imag}(k)}{\partial p} \tag{3.8}$$

in which p represents the various weights. Because the desired motor command u_{nnd} depends only on the controlled system and not on the parameters, (3.8) can be described as

$$\frac{\partial J}{\partial p} = u_{imag}(k) \frac{\partial(-u_{nn}(k))}{\partial p} \tag{3.9}$$

ASSUMPTION II: $u_c(k)$ is approximately proportional to $\lambda u_{imag}(k)$ with $0 < \lambda < 1$.

This assumption can be justified. The (partial) unknown system dynamics can be defined

$$F(\ddot{y}(k), \dot{y}(k), y(k)) = u_c(k) + u_{nn}(k) \tag{3.10}$$

For the optimal set of parameters, (3.10) can be described as

$$F(\ddot{x}(k), \dot{x}(k), x(k)) = u_{nnd}(k) \tag{3.11}$$

These two equations in (3.6)

$$u_{imag}(k) = F(\ddot{x}(k), \dot{x}(k), x(k)) - F(\ddot{y}(k), \dot{y}(k), y(k)) + u_c(k) \quad (3.12)$$

In case of the extended feedback controller, see also section 4, (3.12) can be described

$$u_{imag}(k) = (A + \frac{\partial F(...)}{\partial \ddot{y}(k)})\ddot{e}(k) + (D + \frac{\partial F(...)}{\partial \dot{y}(k)})\dot{e}(k) + (P + \frac{\partial F(...)}{\partial y(k)})e(k) + o(e(k), \dot{e}(k), \ddot{e}(k)) \quad (3.13)$$

The last term is a higher order term in e(k), and hence a small order compared with the linear terms for small e(k). If the feedback gains are chosen sufficiently high, $\lambda \sim 1$. In a following section stability will be considered. With this assumption it would be possible to write (3.5) as

$$J = \frac{1}{2}\lambda^2 {u_c}^T(k) u_c(k) \quad (3.14)$$

where λ reflects the way in which u_c approximates the error between the desired force u_{nnd} and the actual force u_{nn}.

With the assumption II and (3.5) the slope of the performance index can be defined

$$\frac{\partial J}{\partial p} = \lambda u_c(k) \frac{\partial(-u_{nn}(k))}{\partial p} \quad (3.15)$$

$$\frac{\partial J}{\partial p} = -\lambda u_c(k) g_s \sec h^2(v_1(k)) s(k,p) \quad (3.16)$$

where s(k,p) are the signals for the different weights given by

$$s_{ai}(k) = g_s(k) x(k-i) \qquad i = 0,1,2 \quad (3.17)$$

$$s_{bj}(k) = -g_s(k) v_1(k-j) \qquad j = 1,2 \quad (3.18)$$

The gradient of the performance index with respect to the somatic gain g_s is given by

$$\frac{\partial J}{\partial g_s} = \frac{1}{2}\frac{\partial({u_{imag}}^T(k) u_{imag}(k))}{\partial g_s} = -\lambda u_c(k) g_s \sec h^2(v_1(k)) v_1(k) \quad (3.19)$$

The algorithm to modify the parameters can be described by

$$a_i(k) = a_i(k-1) + \alpha_i \lambda u_c(k) g_s \sec h^2(v_1(k)) s_{ai}(k) \qquad i = 0,1,2 \quad (3.20)$$

$$b_j(k) = b_j(k-1) + \alpha_j \lambda u_c(k) g_s \sec h^2(v_1(k)) s_{bj}(k) \qquad j = 1,2 \quad (3.21)$$

$$g_s(k) = g_s(k-1)[1 + \alpha_{gs} \lambda u_c(k) \sec h^2(v_1(k)) v_1(k)] \quad (3.22)$$

where alpha are amplifying terms, the so-called learning rates. The amount of change of the weights when an error is present, is determined by these rates. When the rates are picked too large, this can cause instability. These relations form the algorithm on which the modification of the parameters is based.

4. The Feedback Controller

As already is mentioned in earlier sections, the feedback controller acts as a kind of adaptive critic. The feedback force reflects the quality of the DNU in learning the inverse dynamics of the controlled system. By optimal weights, the neural output equals the desired force and the trajectory error (3.1) tends to zero. In general, the feedback controller can be described by

$$u_c(k) = Pe(k) + D\dot{e}(k) + A\ddot{e}(k) \quad (4.1)$$

The values for the feedback gains have to be chosen more or less arbitrary and they stay fixed in the feedback-error-learning method. In the case of a (partial) known system it is possible to choose these values with an algorithm, e.g. the LQR-routine in Matlab. With optimal chosen values for the feedback gains the feedback force $u_c(k)$ is a linearized model for the inverse of the controlled system and because of that, the feedback force is almost the exact motor command error so the learning will be very fast. The kind of feedback controller that has to be implemented in the control scheme depends on the measurements that can be made on the system output, e.g. is it possible to measure the position, velocity and/or the

acceleration. In other words what information is available on the system output to feed back. When only a P-action is used, the situation of gain explosion can appear. The problem of gain explosion can be solved by reducing the P-value when the error decreases, or that even no motor command error is determined when only a very small trajectory error appears. With only a P-action it is possible to simplify the control scheme by using the error in the trajectory instead of the error in the motor command. This is possible because the relation between $u_c(k)$ and e(k) is relatively simple.

5. Stability

The following function is examined as a candidate of the Liapunov function

$$V = \frac{1}{2} u_{imag}{}^T(k) u_{imag}(k) \geq 0 \qquad (5.1)$$

The function V is only exact zero when the synaptic and somatic weights are optimal and the DNU reflects exact the inverse dynamics of the controlled system. In that case the network motor command equals the desired motor command. Of course this holds for nontrivial x(k). The time derivative of V can be defined

$$\frac{dV}{dt} = u_{imag}{}^T(k) \frac{\partial u_{imag}(k)}{\partial p} \frac{\partial p}{\partial t} + \frac{\partial V}{\partial t} \qquad (5.2)$$

With the equations (3.20)-(3.22) that are derived in earlier sections the derivative of the various parameters with respect to time becomes

$$\frac{\partial p}{\partial t} = p(k) - p(k-1) = -\frac{\partial J}{\partial p} = \frac{\partial u_{nn}}{\partial p} \qquad (5.3)$$

where p(k) is the value of a parameter at time k and p(k-1) the value at time k-1. Using (5.3) it is possible to write (5.2) as

$$\frac{dV}{dt} = \frac{\partial V}{\partial t} + u_{imag}{}^T(k) \frac{d(-u_{nn}(k))}{dp} \left(\frac{d u_{nn}(k)}{dp}\right)^T u_{imag}(k) \qquad (5.3)$$

where the partial derivative of V with respect to time can be defined

$$\frac{\partial V}{\partial t} = \left(\frac{\partial V}{\partial x(k)}\right)\dot{x}(k) + \left(\frac{\partial V}{\partial \dot{x}(k)}\right)\ddot{x}(k) + \left(\frac{\partial V}{\partial \ddot{x}(k)}\right)\dddot{x}(k) \qquad (5.4)$$

So (5.4) is the partial derivative of V with respect to time while keeping the weights constant. When signals like a step signal or a harmonic signal are used for the desired signal x(k), the time average of (5.4) vanishes. This leads to

$$\frac{dV}{dt} \leq 0 \qquad (5.5)$$

From these equations there can be concluded that the parameters p asymptotically converge to the optimal set of synaptic weights. Then, because $u_{imag}(k)$ is zero for optimal weights, y(k) asymptotically converges to x(k).

6. The Control scheme

In this section the control scheme, see figure 3, is considered and explained when necessary.

The controller consists of the elements explained earlier in this article and some unexplained elements. There is a possibility to include disturbances on u(k) to reflect some unknown dynamics of the controlled system. This is done when the so-called *simple system* is considered. The amplifying and scaling terms working on the neural output and desired trajectory are necessary when motor commands larger than one must be applied to the system. This is necessary when large amplitudes for the desired trajectory are used. With large amplitudes the determined motor command is most of the time -1 or +1 unless the somatic gain g_s is very small. Because $u_{nn}(k)$ is defined by

$$u_{nn}(k) = \tanh(v(k)) \qquad (6.1)$$

where the tanh lies between -1 and 1, the amplifying and scaling terms are needed to make larger forces possible to control the system.

7. The simulation results

In this section some simulation results, achieved with Matlab, on the two systems considered are shown. The first part of this section deals with a *simple* mass-spring-damper *system* to

get some idea about the possibilities, problems and the working of the given control scheme. After this, the *system* with the *two masses* is discussed.

7.1 Simulation on a simple system

The *simple system* consists of a mass, a spring and a damper as can be seen in figure 5.

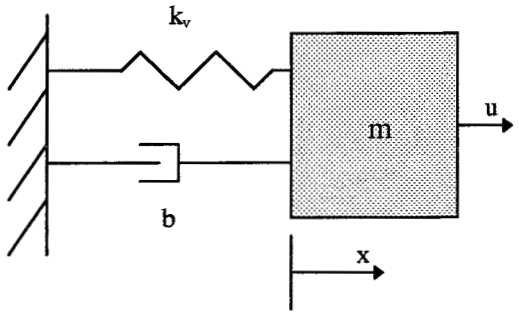


Figure 5 : the simple system

The parameters which represent this elements are chosen before the simulation is started, but can be changed during simulation to see if the DNU stays stable and reduces the trajectory error that appears after a parameter change. To get some idea of the working and the behavior of the controller considered here, a simulation is done with this simple system. Together with a first impression on the values of the weights and the parameters which are reached after learning, this first simulation gives some indication of the length of the learning period. The following equation can be defined for the system

$$m\ddot{y}(k) = u(k) - b\dot{y}(k) - kvy(k) \qquad (7.1)$$

With the LQR algorithm in Matlab the values for a PD-Controller are chosen and characteristic for this controller is that they stay fixed during the whole simulation. These values showed to be of interest especially in the beginning of the simulation, because they determine the applied forces in the beginning. Large values for the parameters lead to large forces.

During the first simulations the various parameters are kept constant, except, of course, the weights of the DNU. In these simulations the input signal is a so-called step signal

$$x(k) = 0.5 \qquad (7.2)$$

In these first simulations the parameters of the system, the mass and the damping and spring constants, are chosen in a way that the applied forces as they appear later during simulation, lie between -1 and +1. This means that the amplifying and scaling term can be chosen as 1. In the resulting plots is found that the neural network reduces the offset of the PD-Controller to zero.

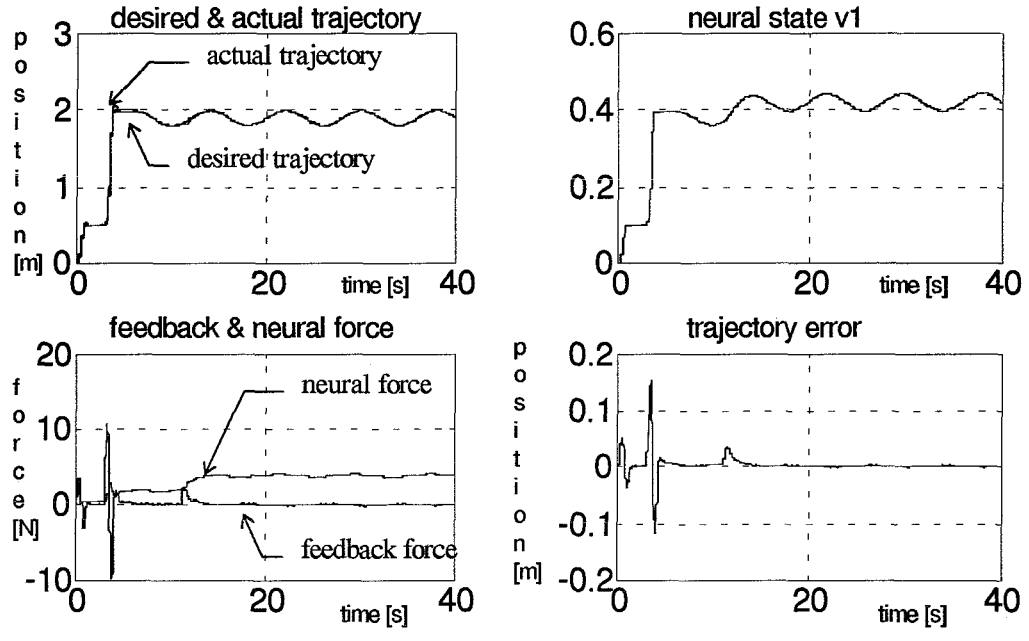
So far, there were no disturbances present on the simulated system. To get a more realistic situation in this simulation process, a disturbance on the applied forces is included. Several disturbances are considered, e.g. one or more sinusoidal signals, a random signal or a combination of these. Also in the simulations with this disturbances the DNU reduces the offset to zero. All the weights have a starting value of 1 and it can be seen that the weights stay in that area. The feedback weights get a little smaller and the feedforward weights a little larger when learning proceeds.

For all simulations done so far, in the beginning the PD-Controller has a large influence on the applied forces, but that influence decreases as the simulation goes on and eventually the DNU defines about 90 % of the applied forces. This is as is expected. However it must be noted that the specific amount of the contribution of the DNU to the applied forces is influenced by the disturbance on u(k). Large disturbances decrease this amount, because the DNU needs time to adjust its parameters and the PD-Controller gains influence.

The DNU seems to stay stable when a parameter change of the simple system appears. Changing the mass has no big influence on the behavior of the neural unit when step signals are considered. Changing the spring constant can make larger forces necessary and the amplifying and scaling terms need to be included.

In all the simulations so far, the input x(k) was a step signal. This causes enormous forces by the PD-Controller and furthermore this desired trajectory is very unrealistic. A possible solution for the appearing large motor commands is to decrease the values for the PD-Controller, but this leads to larger errors when the simulation goes on. Another possibility is to make the input signal a more realistic one. Not an ordinary step signal in which a step is made on one discrete moment but a signal that goes from one point to another in a specific amount of time. In this specific amount of time the desired trajectory can be defined

PLOT 1 : SIMPLE SYSTEM



$$x(k) = c_0 + c_1 dtk + c_2 (dtk)^2 + c_3 (dtk)^3 + c_4 (dtk)^4 + c_5 (dtk)^5 \qquad (7.3)$$

With this signal it takes a second to move from the starting point zero to the desired position 0.5. Doing so decreases the applied forces, which were enormous with the ordinary step signal.

To see what changes and disturbances have a large influence on the behavior of the controller, a simulation is done in which several disturbances are present. So are the values of the mass and the spring constant changed during simulation. The desired trajectory x(k) is a combination of two step signals and a harmonic signal. During the whole simulation there is a disturbance on u(k) which can be described by the following equation

$$0.1\sin(10\,k\,dt) + 0.01\,randn(size(k)) \qquad (7.4)$$

The results of this simulation can be found in plot I and will now be discussed. As can be seen in the resulting plots the learning times are very short, as can be expected for this simple system. Short learning time causes the feedback force to decrease very quickly and the neural force dominates almost immediately. When a new signal is defined as desired trajectory, the neural force starts to dominate again after a small learning period. The trajectory error is small during the simulation especially after the learning period. Besides changing the desired trajectory also the parameters of the simple system are changed. Especially the changing of the spring constant calls for larger needed forces and the amplifying and scaling terms need to be included. In case of the ordinary PD-Controller, which is used as a reference, the trajectory error changes when a new input signal is applied. Desired trajectories with higher amplitude lead to larger trajectory errors with just a PD-controller and no neural network included. Because of the disturbances on u(k), the nonlinear part of this simulation, the PD-part of the controller keeps to play a role in the whole control scheme. When the influence of this disturbance is decreased, the feedback force decreases also.

Now there will be a closer look at the *two mass system* with friction. These nonlinear friction terms are shown in figure 6. The simulations with this system to be controlled are more realistic because it is a model of the system that is used in the experiments done with this controller as can be found in Section 8.

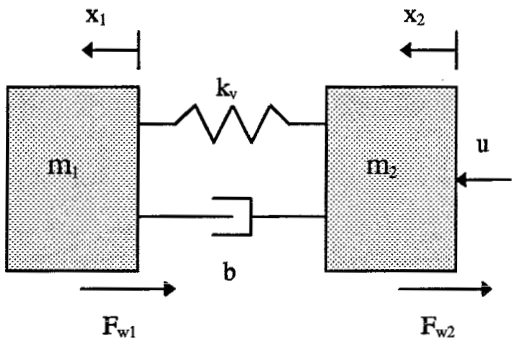


Figure 7 : two masses with friction

The spring constant is rather large compared to the damping constant and two masses. The desired trajectory for mass 1 of this system is a sine defined by

$$x(k) = 0.4\sin(k\,dt) \tag{7.5}$$

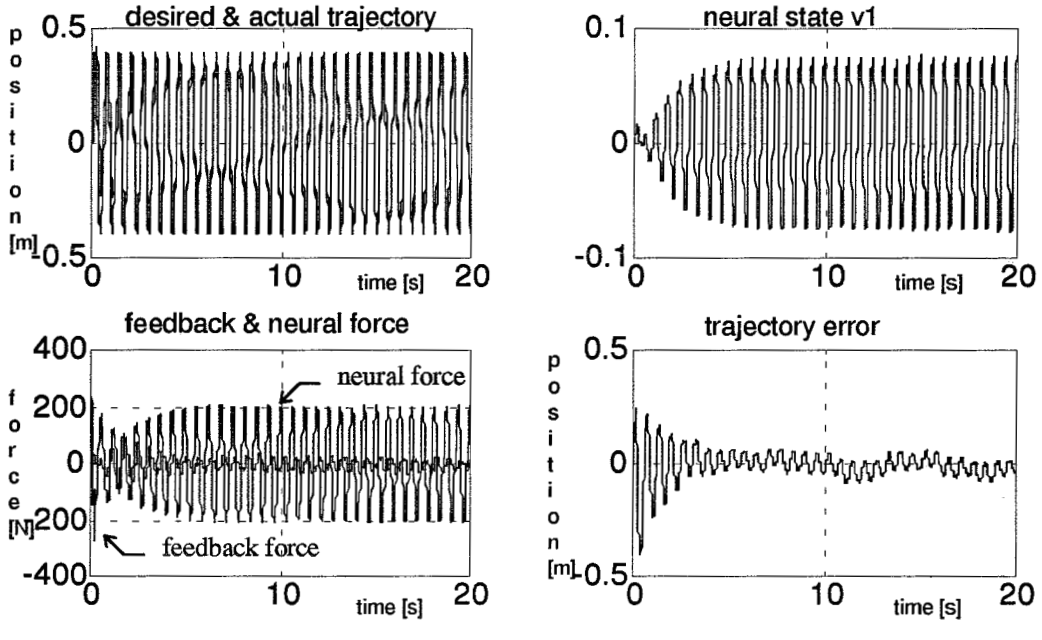
Compared to the *simple system* there are a few important differences. Moving the first mass leads to a transfer of the second mass and when the desired positions are reached no force is necessary to keep the masses at the desired place. The masses are not connected to the real world by a spring and/or damper as was the case in the first simulations. There, a force is needed to keep the mass at the desired position, unless this position is the equilibrium. A second difference is the fact that nonlinear terms are involved in the state equations. This nonlinear terms are of the form

$$F_d = -C\,sign(velocity) \tag{7.6}$$

where C is the friction coefficient. On both masses there is a nonlinear term of the former kind, with different friction coefficients.

The resulting plots show that the learning time compared to the learning time as occurred in plot 1 has been increased. This is caused by the more complicated system where nonlinear terms are involved, other than disturbances on the applied force u(k). The increase in learning time makes it impossible to use different signals in one desired trajectory as is done in the first simulations (plot 1). This because of the length of the learning time. While the neural network is learning, the feedback force and the trajectory error both start to decrease and the neural force gains importance. After a while the neural force dominates the applied force u(k), as can be seen in the resulting plots. After the learning period the various weights reach a constant value with only very small fluctuations. These plots are not included because they are of minor importance.

PLOT 2 : TWO MASSES



8. Experiments

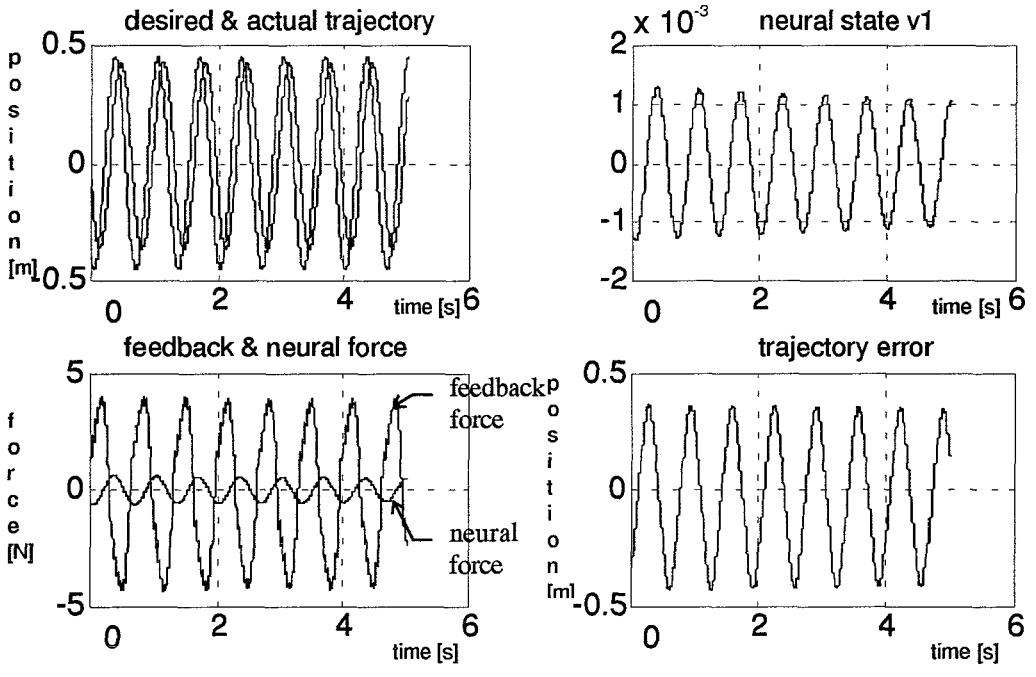
A system consisting of two rotating masses is used in the experiments which is controlled by the control scheme considered in this article, see figure 3. Although the masses in the experiments can rotate, it is interesting to compare the results with the simulations on the *two mass system* with friction, which is a simplified model of the two rotating masses. In the experiments no step signals are used because of the long learning time, according the simulations done with the *two mass system*. Possible knowledge on the friction or other system parameters is not used in the control of the system.

The implementation of the control scheme is rather complicated in comparison with the simulations. For the real-time implementation of the control scheme C++ is used. The control scheme for the experiment is created in Simulink, within Matlab. Simulink contains a special written C-file where the several weights are updated. The computer is linked with the real system on which the experiments are running. An encoder measures the position reached by the mass and this measurement is used in the programming to compute the next motor command.

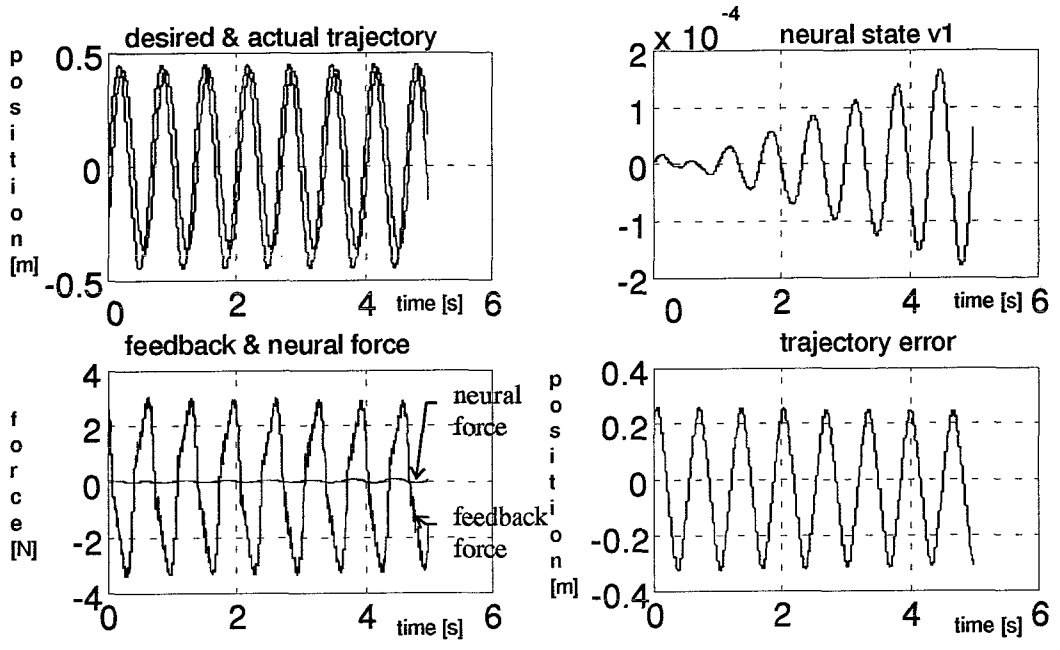
Now the results of the experiment, as can be found in plot 3, 4 & 5 will be discussed and compared to the simulation results as given in the previous section. The plots show the situation at the initial stage, the intermediate stage and the final stage.

The time mentioned in the plots 3, 4 & 5 is not the absolute time, but the time after starting that particular data storage. In the experiments the learning time is considerably increased compared to the simulations with the *two mass system*. This can be explained by nonlinearity's that play a role in the experiments and which are not modeled in the simulation model where only a simple model of the friction is included. The neural state $v_1(k)$ is plotted to illustrate the length of the learning period. After the experiment is started the neural state $v_1(k)$ follows the desired trajectory x(k) (plot 3). When learning proceeds the neural state changes (plot 4) and the neural force starts to dominate (plot 5), as is expected. This process also occurs in the simulations. In the experiment there is an offset visible which is not found in the simulations. It seems that the friction is not equal in both directions. This is according other experiments that have been carried out on this system with the two rotating masses. Plot 3 seems to show a difference in phase between the desired and actual trajectory. It is possible to increase the feedback force, but after the intermediate stage (plot 4) the difference decreases and in the final stage (plot 5) there is no difference in phase between the trajectories.

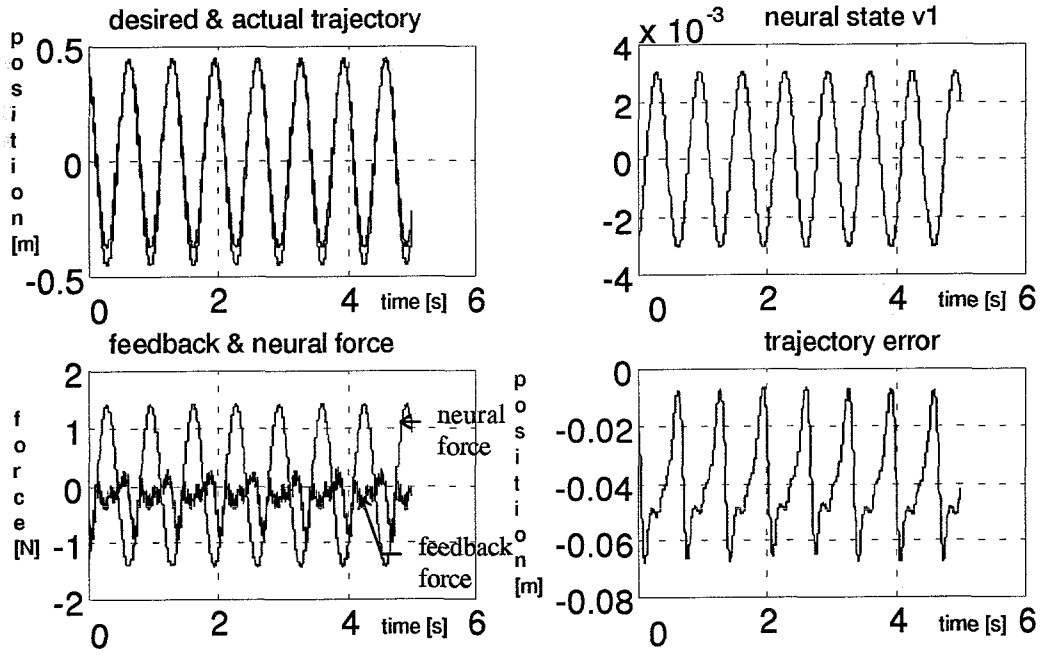
PLOT 3 : EXPERIMENT, INITIAL STAGE



PLOT 4 : EXPERIMENT, INTERMEDIATE STAGE



PLOT 5 : EXPERIMENT, FINAL STAGE



The learning time is very sensitive to the starting values of the various weights. When the parameters are chosen in the area of the values they reach by learning compared to more arbitrarily chosen values, the learning will be reduced. Of course, when no information on the system to be controlled is available, a few experiments have to be carried out to know which values are reached by the various parameters. The learning rates are very important for the behavior of the control scheme. Large learning rates cause instability and small learning rates cause long learning times.

9. Conclusions

In several simulations and experiments it was shown that the neural unit learns and that it takes over the control of the (partially) unknown system. So the feedback force is reduced as is expected in the beginning of this article. The various weights reach a constant value with only very small fluctuations. The neural network stays stable during the experiments. The total error can possibly made smaller after some tuning operations are executed. The offset visible in the experiments can possibly be reduced by adding an integration action to the feedback controller. It is also possible to use more layers and/or input units in the neural network as are used in the control scheme proposed here. The neural network as it is used in the simulations and experiments includes only one unit. This small network shows the working of the unit, but the neural network probably must be enlarged to achieve better results.

The following elements could be part of future research. Research on methods to tune the network, to make it more easy to choose the right values for the parameters and the weights. Tuning can make the learning act faster and can improve the behavior of the controller. The scaling and amplifying terms need to be investigated in more detail so large forces can be applied to the system to be controlled. Also the judging of the quality λ (assumption II) needs to be explored in future research. An appropriate feedback controller can fasten the learning considerably. Because the plant in this control scheme has only minor nonlinearity's, other methods to control this plant can probably give better results. The goal of this experiment was to see whether the proposed controller is capable in controlling (partial) unknown plants. To show the capabilities of the proposed controller, experiments with more complicated systems are necessary.

References

[1] Gupta, M.M. and Rao, D.H. (1993). Dynamic Neural Units with Applications to the Control of Unknown Nonlinear Systems. *Journal of Intelligent and Fuzzy Systems*, Vol. 1 (1), 73-92.

[2] Kawato, M. (1990). Feedback-Error-Learning Neural Network for supervised Motor Learning. *Advanced Neural Computers*, Elsevier Science Publishers B.V., 365-372.

[3] Mendel, J.M. and McLaren, R.W. (1970). Reinforcement learning control and pattern recognition systems. *Adaptive, learning and pattern recognition systems: Theory and applications*, Academic Press, 287-318.

[4] Rao, D.H. and Gupta, M.M. (1993). Dynamic Neural Adaptive Control Schemes for Linear and Nonlinear Systems. *Proceedings of the American Control Conference San Francisco, California, June 1993*, 1450-1454.