The 1998 IEEE International Joint Conference on Neural Networks Proceedings

(V. 3) (V. B)

The 1998 IEEE International Joint Conference on Neural Networks Proceedings

IEEE World Congress on Computational Intelligence

May 4 - May 9, 1998
Anchorage, Alaska, USA

Volume 3 Page 1688 - 2561

The 1998 IEEE International Joint Conference on Neural Networks Proceedings

IEEE Catalog Number 98CH36227
ISBN 0-7803-4859-1 (softbound)
ISBN 0-7803-4860-5 (casebound)
ISBN 0-7803-4861-3 (microfiche)
ISBN 0-7803-4862-1 (CD-ROM)
ISSN 1098-7576

Additional copies of this publication are available from

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
1-800-678-IEEE or 1-732-981-0060

Coordinated and produced by

ALTEC
Technology Driven Print Solutions

in cooperation with

DOCUMATION

Contact ALTEC at 800.997.9921 x 1334

Author's Index

M

N

O

P

X

Y

Z

PID Controller Design Using Dynamical Neural Networks

Wen-Shyong Yu, Tatung Institute of Technology, Taipei, Taiwan, R.O.C.
Tien-Ching Lu, Tatung Institute of Technology, Taipei, Taiwan, R.O.C.

1 abstract

In this paper, a new PID control scheme using neural networks is presented. The scheme is to parameterize a Ziegler-Nichols-like formula with a single parameter and use the dynamical neural networks to tune the parameter. A Lyapunov type stability criterion is derived to ensure the convergence of the neural procedure and guarantee the stability of the closed-loop system. The simulation examples with small and large time delay are given to illustrate the performance of the proposed method. Simulation results demonstrate that better control performance can be achieved when compared with that of the Ziegler-Nichols PID controllers and that of the fuzzy selftuning of PID control scheme.

2 Introduction

Despite the appearance of many sophisticated control theories and techniques, the majority of industrial processes nowaday still use the proportional-integral-derivative (PID) controllers, since their simple structure and easy implementation. However, a crucial issue for designing the PID controllers is how to set the control parameters. The conventional way to set them is by try and error method. It has been shown that the PID control scheme is adequate to achieve desired performance if the process is modeled perfectly and the load is not changed frequently. However, a real industrial process can never have such ideal peoperties and fixed loads. So far, great effort has been devoted to developing methods for choosing the optimizing control parameters, especially for dealing with time-delay processes [1, 2]. In general, the control parameters of the PID controllers are usually selected to be fixed during control when they are tuned or chosen in a certain optimal way. However, such method cannot obtain effectively control performance since the control parameters can not changed when the load disturbances change. Thus, it may always need frequently on-line re-tune in order to obtain good performance [3, 4, 5]. In this paper, a new PID control scheme using neural networks is presented. The scheme parameterizes the Ziegler-Nichols-like formula with a single control parameter and exploits the possibilities of the controllers for achieving better performance for the unknown systems with uncertainty or known structures with unknown variation in parameters due to the load disturbances. The control parameters is recursively learning from the error signal between the plant output and the reference input. The simulation examples with small and large time lags are given to illustrate the performance of the proposed method. From simulation results, better control performance can be expected by the proposed method than that of the PID controllers or that of the fuzzy selftuning of PID control scheme.

3 Problem Formulation

To begin with, we assume that the process to be controlled has single input $u(t)$ and single

0-7803-4859-1/98 $10.00©1998 IEEE

output $y(t)$, and the control objective is to let the process output follow a prescribed set-point y_r. Let the tracking error be defined by

$$e(t) = y_r - y(t) \quad (1)$$

The PID controller, which generates the control signal, $u(t)$, has the following:

$$u(t)=K_c\left[e(t)+T_d\frac{de(t)}{dt}+\frac{1}{T}\int e(t)dt\right] \quad (2)$$

where K_c,T_d,T_i are three control parameters and are defined, respectively, as the proportional gain, the derivative time and the integral time of the controller. From [4], the control parameters can be parameterized by a single parameter α and are shown in the following

$$\begin{cases} K_c &= 1.2\alpha k_u \\ T_i &= 0.75\frac{1}{1+\alpha}t_u \\ T_d &= 0.25T_i \end{cases} \quad (3)$$

where k_u and t_u are the ultimate gain and the ultimate period of the underlying process respectively. The form of the parameterization is inspired by the Ziegler-Nichols formula when $\alpha = 0.5$. Thus, we can think of the controller as to improve the Ziegler-Nichols control scheme by biasing all the control parameters in order to control the process output to a prescribed signal.

In this paper, the $\alpha(t)$ is defined by the following dynamical equation

$$\dot{\alpha} = -\beta\alpha + \omega s(\alpha) \quad (4)$$

where β is a positive constant and ω is a weight. Let $s(\alpha)$ be a smooth (at least twice differentiable), monotone increasing function and be represented by sigmoids of the form

$$s(\alpha) = \frac{k}{1+e^{l\alpha}} \quad (5)$$

where k, l are, respectively, parameters representing the bound and slop of sigmoid's curvature [6].

The purpose of the paper is to find the learning rule for weight, ω, such that the closed-loop system is stable and weight can converge to its true weight value, ω^*. We assume that ω^* exists, then, in the steady-state, (4) becomes

$$\dot{\alpha} = -\beta\alpha + \omega^* s \quad (6)$$

Let

$$h = \frac{1}{2}\alpha^2 \quad (7)$$

then

$$\begin{aligned} \dot{h} &= \frac{dh}{d\alpha}\dot{\alpha} \\ &= \alpha(-\beta\alpha + \omega^* s) \end{aligned} \quad (8)$$

Define

$$\dot{\eta} + \gamma\eta = -\dot{h} + \alpha(-\beta\alpha + \omega s) \quad (9)$$

where γ is some positive constant. Let the Lyapunov function candidate be defined by

$$V = \frac{1}{2}\eta^2 + \frac{1}{2}\tilde{\omega}^2 \quad (10)$$

where $\tilde{\omega} = \omega - \omega^2$. Differentiating (10) and using (9), we obtain

$$\dot{V} = -\gamma\eta^2 + \eta\alpha\tilde{\omega}s + \tilde{\omega}\dot{\tilde{\omega}} \quad (11)$$

If we choose

$$\dot{\tilde{\omega}} = \dot{\omega} = -\alpha\eta s \quad (12)$$

then (11) becomes

$$\dot{V} = -\gamma\eta^2 \quad (13)$$

Now we can prove the following theorem.

Theorem 1: Consider the Lyapunov function (10). The learning rule

$$\dot{\omega} = -\alpha\eta s \quad (14)$$

guarantees the following properties

- $\eta, \alpha, \tilde{\omega} \in L_\infty$
- $|\eta| \in L_2$
- $lim_{t\to\infty}\eta = 0$
- $lim_{t\to\infty}\dot{\omega} = 0$

Proof: From (11) and (14), $\dot{V}$ becomes

$$\dot{V} = -\gamma\eta^2 < 0$$

Hence, $V \in L_\infty$, which implies $\eta, \tilde{\omega} \in L_\infty$. Since V is a nonincreasing function of time and is bounded from below, $lim_{t\to\infty}V = V_\infty$ exists. Therefore by integrating $\dot{V}$ from 0 to ∞, we have

$$\int_0^\infty \|\eta\|^2 dt = \frac{1}{\gamma}[V(0) - V(\infty)] < \infty$$

which implies that $\eta \in L_2$. By definition, the sigmoid function $s(\alpha)$ is bounded for all α and, by assumption, all inputs to the dynamical adaptation are also bounded. Hence, from (9), we have $\dot{\eta} \in L_\infty$. Since $\eta \in L_2 \bigcap L_\infty$ and $\dot{\eta} \in L_\infty$, using Barbalat's Lemma, we conclude that $lim_{t\to\infty}\eta = 0$. Now using the boundedness of $s(\alpha)$ and the convergence of η to zero, we have that $\dot{\tilde{\omega}}$ also converges to zero.

4 Example

Let the controlled process with time delay be defined by

$$G(s) = \frac{k_p e^{-\theta_d s}}{(Ts+1)^2} \tag{15}$$

The parameters k_p and T are all set to 1, and the dead-time, θ_d, will be set to two different values to examine the ability of the proposed control scheme in handling small or large dead-time. First, the value of θ_d is chosen to be 0.3 and y_r is set to be 1 to track a unit step signal, and the load disturbance, $0.2sin(t)$, is introduced when $t = 20$. The simulation results are shown in *Figure* **1** - *Figure* **4** . *Figure* **3** shows the performance of the PID controller with dynamical neural network for the set-point response which is better than that of the other two controllers shown in *Figure* **1** and *Figure* **2** with shorter rise time, shorter setting time, and less overshoot. Observe that unlike other controllers where lowing the overshoot is often at the expense of slowing down considerably the rise time, the proposed control scheme seems to reconcile these two requirements. The reason is that the dynamical neural type PID controller allows the selection of different control parameters for controlling the process at every time interval. The controller is then designed so that it can pick up an appropriate performance at every control stage. Next, the value of θ_d is then set to be 0.8. The load disturbance, $0.2sin(t)$, is also introduced at $t = 20$ to allow the process to settle down. The simulation results are shown in *Figure* **5** - *Figure* **8** . From *Figure* **1** - *Figure* **3** and *Figure* **5** - *Figure* **7** , we can conclude that the set-point response improves considerably while the responses are either comparable to those obtained by the conventional controllers (in the case of small dead-time), or only improves marginally (in the case of large dead-time).

5 Conclusions

In this paper, We have presented a detailed account of the PID controller using dynamical neural networks. One interesting feature of the present work is the combination of the conventional PID controller with the dynamical neural networks. Because of our scheme's connection with Ziegler-Nichols formula, the proposed control scheme is not completely model-free. We have provided a method to derive a learning rule for the weight of the neural networks such that the control parameters of the PID controller can be tuned properly whenever the load

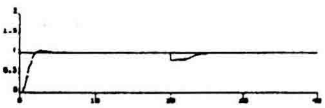

Figure 1: The response of the PID controller using the Ziegler- Nichols formula, where $K_c = 2.808, T_I = 1.64, T_d = 0.41$.

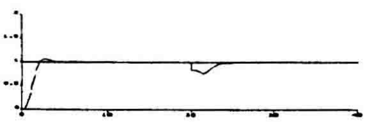

Figure 2: The response of the fuzzy self-tuning PID controller.

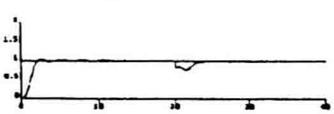

Figure 3: The response of the PID controller with dynamical neural networks.

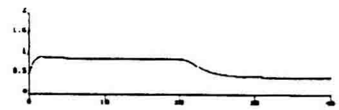

Figure 4: The parameter α.

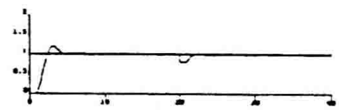

Figure 5: The response of the PID controller using the Ziegler- Nichols formula, where $K_c = 1.8, T_i = 2.4, T_d = 0.6$.

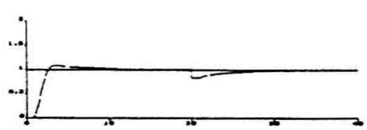

Figure 6: The response of the fuzzy self-tuning PID controller.

disturbances exist. From simulation results, the performance of the proposed control scheme is compared favorably to those of the PID control scheme and the fuzzy self-tuning of PID control scheme.

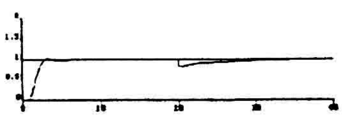

Figure 7: The response of the PID controller with dynamical neural networks.

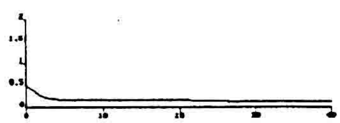

Figure 8: The parameter α.

References

[1] Hang, K. J. A°strom, and W. K. Ho, "Refinements of the Ziegler-Nichols tuning formula," Proc. IEE, Pt.D., vol. 138, pp. 111-118, 1991.

[2] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatical controllers," Trans. ASME, vol. 64, pp. 759-768, 1942.

[3] P. J. Gawthrop and P. E. Nomikos, "Automatic tuning of commercial PID controllers for single-loop and multiloop applications," IEEE Control Syst. Mag., vol. 10, pp.34-42, 1990.

[4] S. Z. He, S. Tan, and F. L. Xu, "Fuzzy self-tuning of PID controllers," Fuzzy Sets Syst., vol.56, pp. 37-46, 1993.

[5] Z. Y. Zhao, M. Tomizuka, and S. Isaka, "Fuzzy Gain Scheduling of PID Controllers," IEEE Trans. Syst. Man, Cybern., vol. 23, no. 5, pp. 1392-1398, 1993.

[6] G. A. Rovithakis, and M. A. Christodoulou, "Adaptive Control of Unknown Plants Using Dynamical Neural Networks," IEEE Trans. Syst. Man, Cybern., vol 24, no. 3, pp. 400- 412, 1993.

[7] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Feedback Control of Dynamic Systems, 3rd ed. Menlo Park,CA: Addison-Wesley, 1994.

[8] C. T. Chen, Analog and Digital Control System Design: Transfer-Function, State-Space, and Algebraic Methods, Saunders College Publishing, 1993.

Control of a Static Nonlinear Plant Using a Neural Network Linearization

Jürgen Van Gorp, Vrije Universiteit Brussel, Brussel, Belgium

Abstract **— One possibility to control a static plant is the design of a controller based on the inverse of an identified model. For nonlinear plants, determining or identifying the plant model may be a difficult task. When a state space model of the plant is not explicitly needed, it is possible to consider the plant as a black box and approximate the plant using neural networks. In this paper a control strategy is presented, based on the combination of classical linear control methods with a neural network that inverses the plants nonlinear characteristics. A proof is given that the plant can be positioned with an arbitrary small positioning error. The method is experimentally illustrated on the positioning control of a flexible robot arm. The results of the neural network based control are compared with a PI controller.**

1. Introduction

In the past few years, much effort has been put into the control of nonlinear robot arms. In general these arms are considered as flexible beams, for which the dynamics are identified, and an inverting controller is developed (see [6] ... [8]). In our problem, the manipulator kinematics are strongly nonlinear, such that the kinematics could not easily be identified. The goal here is to find a control force that can position the spring manipulator accurately to any point within a given work space, called the *region of interest.*

In a first attempt to solve the identification problem, a nonlinear model of the plant was constructed. The spring was modelled as a chain of flexible links. Benati [1] describes the equations of motion for such a chain. The same is done by Oliviers and Campion [9], with the remaining task of identifying the many parameters and finding the derivation of the inverse kinematics needed for control. In a more elaborate case where the manipulator has unresolved kinematics, it can become redundant [5] or hyper-redundant [2]. Hirose states in [5] that even in the redundant case, it is impossible to derive the inverse Jacobian matrix of the kinematics and dynamics.

A study on a nonlinear plant based on a pneumatic driven manipulator, was recently done by Hesselroth [4] . He put both the controller and the linearization scheme in one single black box.

This paper considers only the plant as a black box. Measurements are used to train an inversing neural network [10] off-line to linearize the plant. Next, a controller is designed using classical PI control. Section 2. of this paper will give the assumptions under which the control can be mathematically described. Section 3. will give a practical example on the use of the control scheme.

The author is with the V.U.B., Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussel, Belgium, E-mail: jvgorp@vub.ac.be

2. Linearization using Neural Networks

2.1. Assumptions

Consider a static nonlinear plant h that maps an input $U \in \Re^2$ to the output $Y \in \Re^2$ with U and Y considered as column vectors: $U = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$ and $Y = \begin{bmatrix} y_1 & y_2 \end{bmatrix}^T$ such that $y_i = h_i(U) = h_i(u_1, u_2)$; $i = 1, 2$.

Assumption 1: The plant h is locally linearizable in each point $U_0 \in \Re^2$. Higher order derivatives of h are negligible compared to the first order derivative. This means that the plant is continuous and for any point U in a neighbourhood of U_0 yields:

$$h_i(U) \cong h_i(U_0) + (\Delta U)^T \frac{\partial h_i(U_0)}{\partial U} \qquad i = 1, 2 \qquad (1)$$

with $\Delta U = U - U_0$ and $\frac{\partial h_i(U)}{\partial U} = \left[\frac{\partial h_i(U)}{\partial u_1} \;\; \frac{\partial h_i(U)}{\partial u_2}\right]^T$

Assumption 2: For each $U \in \Omega$ with $\Omega \subset \Re^2$ corresponding to the region of interest of the robot arm, the first derivative of $h_i(U)$ exists and:

$$0 < \left|\frac{\partial h_i(U)}{\partial U}\right| < \infty \qquad i = 1, 2 \qquad (2)$$

Denote $\Psi \subset \Re^2$ as the space of outputs related to Ω such that $Y \in \Psi$ and $U = h^{-1}(Y) = \begin{bmatrix} h_1^{-1}(y_1, y_2) \\ h_2^{-1}(y_1, y_2) \end{bmatrix}$

Based on a series of positioning measurements, a number of input-output pairs $(U_i, Y_i); i = 1...N$ is taken. The inputs and outputs are restricted to the area of interest of the robot arm and are used to train a neural network to form a mapping $\tilde{h}^{-1}$ that inverses the relation $Y = h(U)$:

$$\tilde{h}^{-1}(Y): \Psi \to \Omega: \tilde{U} = \tilde{h}^{-1}(Y) = \begin{bmatrix} \tilde{h}_1^{-1}(y_1, y_2) \\ \tilde{h}_2^{-1}(y_1, y_2) \end{bmatrix} \qquad (3)$$

Assumption 3: This inversion is not perfect and differs from the real inverting function $h^{-1}(Y)$ with an error $\delta h^{-1}(Y)$ that can be added to the real inverting function:

$$\tilde{h}_i^{-1}(Y) = h_i^{-1}(Y) + \delta h_i^{-1}(Y) \qquad i = 1, 2 \qquad (4)$$

0-7803-4859-1/98 $10.00©1998 IEEE

Assumption 4: The errors $\delta h_i^{-1}(Y)$ have an upper limit $\overline{\delta h^{-1}}$ such that for all $Y \in \Psi$ yields

$$\overline{\delta h^{-1}} = \max_{\substack{Y \in \Psi \\ i = 1,2}} |\delta h_i^{-1}(Y)| \qquad (5)$$

In practise $\overline{\delta h^{-1}}$ is determined from a validation set of input-output pairs $(U_j, Y_j); j = 1...M$ that are also taken from the area of interest.

Assumption 5: For the neural network approximation $\tilde{h}^{-1}(Y)$, a class of neural network functions is used that are differentiable with respect to Y such that the first derivative $\frac{\partial \tilde{h}^{-1}(Y)}{\partial Y}$ can be calculated for all $Y \in \Psi$. This excludes the use of e.g. hard limit transfer functions or saturated linear transfer functions [3].

Assumption 6: The identification of the inverse of the plant $\tilde{h}^{-1}(Y)$ is done in such a way that within the area of interest, variations on the error are negligible with respect to variations on $h^{-1}(Y)$. The derivation of equation (4) then gives:

$$\forall Y \in \Psi; \frac{\partial \tilde{h}^{-1}(Y)}{\partial Y} \cong \frac{\partial h^{-1}(Y)}{\partial Y} \qquad (6)$$

or, also: for each Y in a neighbourhood of $\hat{Y}$ yields:

$$\delta h^{-1}(\hat{Y}) \cong \delta h^{-1}(Y) \qquad (7)$$

This assumption demands that the neural network does not only fit the surface, but also the slope of the surface at any point. This is the case as long as the neural network is not overparametrized.

Corollary 1: $h^{-1}(Y)$ and $\tilde{h}^{-1}(Y)$ are locally linear.

Proof: According to assumptions 1 and 2, $Y = h(U)$ is locally linearizable with finite, nonzero derivatives. Then also $U = h^{-1}(Y)$ is locally linearizable and from assumption 6 follows that $\tilde{h}^{-1}(Y)$ can be locally linearized. □

Lemma 1: For an input $E = [e_1 \quad e_2]^T$ the concatenation of the transfer functions $h(\tilde{h}^{-1}(E))$ can be written as

$$h_i[\tilde{h}^{-1}(E)] = e_i + \sum_{j=1,2} \delta h_j^{-1}(E) \frac{1}{\frac{\partial \tilde{h}_j^{-1}(E)}{\partial y_i}} \qquad i = 1,2 \quad (8)$$

Proof: With assumption 3, $h_i[\tilde{h}^{-1}(E)]$ can be written as

$$\begin{aligned} h_i[\tilde{h}^{-1}(E)] &= h_i(\tilde{h}_1^{-1}(E), \tilde{h}_2^{-1}(E)) \\ &= h_i(h_1^{-1}(E) + \delta h_1^{-1}(E), h_2^{-1}(E) + \delta h_2^{-1}(E)) \end{aligned} \qquad (9)$$

and by assumption 1

$$h_i[\tilde{h}^{-1}(E)] = h_i(h_1^{-1}(E), h_2^{-1}(E)) + \sum_{j=1,2} \delta h_j^{-1}(E) \frac{\partial h_i(h_1^{-1}(E), h_2^{-1}(E))}{\partial u_j} \qquad (10)$$

corollary 1 states that h is locally linear and its derivative is finite (assumption 2). Therefore the derivative of the inverse of h exists and is nonzero. This gives:

$$h_i[\tilde{h}^{-1}(E)] = h_i(h_1^{-1}(E), h_2^{-1}(E)) + \sum_{j=1,2} \delta h_j^{-1}(E) \frac{1}{\frac{\partial h_j^{-1}(E)}{\partial y_i}} \qquad (11)$$

and by assumption 6

$$h_i[\tilde{h}^{-1}(E)] = e_i + \sum_{j=1,2} \delta h_j^{-1}(E) \frac{1}{\frac{\partial \tilde{h}_j^{-1}(E)}{\partial y_i}} \qquad (12)$$

with $i = 1, 2$ □

2.2. Design of a Feedforward Control Scheme

An intuitive way to use the inverting function on the plant is to put it in sandwich between a linear controller G and the plant. Therefore consider the setup of Fig. 1.

The controller $G(t): \Re^2 \rightarrow \Re^2 : U(t) = G(t) * \varepsilon(t)$, with $*$ the convolution in time, is not directly connected to the plant but separated by a feedforward linearizing function.

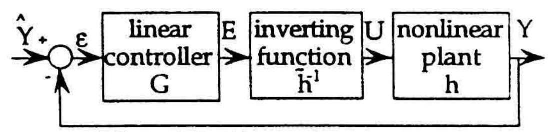

Figure 1. feedforward control scheme

This control scheme differs from many other neural network control schemes in the fact that the controller itself is not included in the neural network, as is the case in e.g. [4] and [11]. The closed loop relation is:

$$Y(t) = h(\tilde{h}^{-1}(G(t) * [\hat{Y} - Y(t)])) \qquad (13)$$

Assumption 7: The bandwidth of $[\hat{Y} - Y(t)]$ is very small compared to that of the controller G such that $G(t) * [\hat{Y} - Y(t)]$ can be approximated by a plain multiplication $G(t)[\hat{Y} - Y(t)]$. For this, both the bandwidths of the setpoint $\hat{Y}$ and the plant h are expected to be very small compared to the bandwidth of the controller G. This assumption is based upon the observation that in our case the robot arm moves very slowly and is heavily damped.

Hence, equation (13) can be written as:

$$Y(t) \cong h(\tilde{h}^{-1}(G(t)[\hat{Y} - Y(t)])) \qquad (14)$$

in which G decouples the control of $h_1(E)$ and $h_2(E)$:

$$G(t) = \begin{bmatrix} g_{1,1}(t) & 0 \\ 0 & g_{2,2}(t) \end{bmatrix} = \begin{bmatrix} g_1(t) \\ g_2(t) \end{bmatrix} \quad (15)$$

Applying Lemma 1 on equation (14) gives:

$$y_i(t) = e_i + \sum_{j=1,2} \delta h_j^{-1}(E(t)) \frac{1}{\frac{\partial \bar{h}_j^{-1}(E(t))}{\partial y_i}} \quad i = 1, 2 \quad (16)$$

$$= \frac{g_{i,i}(t)}{1 + g_{i,i}(t)} \hat{y}_i(t) + \Delta y_i(t)$$

with $E(t)$ defined as $E(t) = G(t)(\hat{Y} - Y(t))$. It can be shown that the error $\Delta y_i(t)$ becomes a complex function in which both $\delta h_i^{-1}(Y)$ and $\frac{\partial \bar{h}_j^{-1}(Y)}{\partial y_i}$ are applied on $E(t)$. The possibility exists that $E(t) = G(t)(\hat{Y} - Y(t)) \notin \Psi$, hence the neural network is excited outside the region of interest. The results of this extrapolation of the neural network mapping are uncertain and may lead to instabilities or uncontrolled modes on the output U. Therefore, an alternative setup is considered in the next section.

2.3. Design of a Feedback Control Scheme

An alternative setup is shown in Fig. 2. The linear controller $G(t): \Re^2 \to \Re^2 : U(t) = G(t) * \varepsilon(t)$ is directly connected to the nonlinear plant and the inverting function is put into the feedback loop. The setpoint $\hat{Y}$ is inverted using the same neural network to become a corrected setpoint $\hat{\kappa}$.

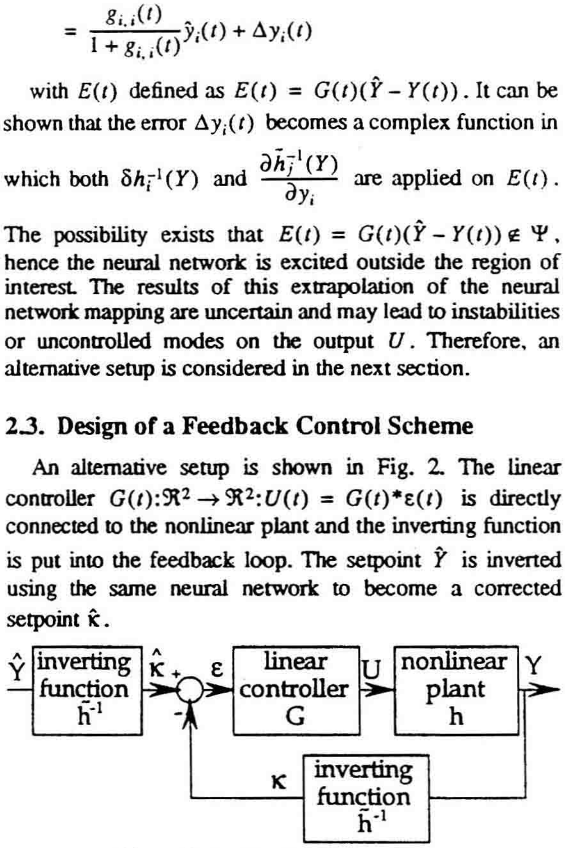

Figure 2. **feedback control scheme**

$G(t)$ is determined from the closed loop relation:

$$Y(t) = h(G(t) * [\bar{h}^{-1}(\hat{Y}) - \bar{h}^{-1}(Y(t))]) \quad (17)$$

With assumption 7 this becomes:

$$h_i^{-1}(Y(t)) = g_{i,i}(t)[\bar{h}_i^{-1}(\hat{Y}) - \bar{h}_i^{-1}(Y(t))] \quad i = 1, 2 \quad (18)$$

$$g_{i,i}(t) = \frac{h_i^{-1}(Y(t))}{\bar{h}_i^{-1}(\hat{Y}) - \bar{h}_i^{-1}(Y(t))} \quad i = 1, 2 \quad (19)$$

2.4. Feedback control scheme convergence analysis

Lemma 2: It is always possible to control the output $Y(t)$ of the nonlinear plant in Fig. 2 to any desired state $\hat{Y} \in \Psi$ with a maximum error

$$|\Delta \hat{y}_i(t)| \le \overline{\delta h^{-1}} \sum_{j=1,2} \left| \frac{1}{\frac{\partial \bar{h}_j^{-1}(\hat{Y})}{\partial y_i}} \right| \quad i = 1, 2 \quad (20)$$

Proof: Break the closing loop in Fig. 2 and choose $g_{i,i} = 1$. The remaining network is a feed-forward control with output

$$Y(t) = h(\bar{h}^{-1}(\hat{Y})) \quad (21)$$

Using Lemma 1 gives

$$h_i(\bar{h}^{-1}(\hat{Y})) = \hat{y}_i + \sum_{j=1,2} \delta h^{-1}(\hat{Y}) \frac{1}{\frac{\partial \bar{h}_j^{-1}(\hat{Y})}{\partial y_i}} \quad i = 1, 2 \quad (22)$$

$$= \hat{y}_i + \Delta \hat{y}_i(t)$$

Following assumption 4 it is possible to find an upper bound for the error δh^{-1}, giving the inequality of equation (20). □

Lemma 3: The nonlinear plant in Figure 2. can be globally controlled to any $\hat{Y} \in \Psi$ with an arbitrary small error.

Proof: Following Lemma 2 the output $Y(t)$ of the plant can be controlled to within a neighbourhood of $\hat{Y}$ as given in equation (20). At that point the feedback loop is closed and the gain G of the controller is taken as in equation (19).

Use corollary 1 on equation (19) and replace the exact inverting function by its neural network approximation. Then it can be proven that the discrete time update G^k at every time step k, can be calculated as :

$$g_{i,i}^k \ge \left| \frac{\bar{h}_i^{-1}(Y^k)}{\overline{\delta h^{-1}} M} \right| \quad (23)$$

This will lead us to the following assumption:

Assumption 8: $g_{i,i}(t)$ is assumed to be much larger than 1 at any time during the control, or $g_{i,i}^k \gg 1$.

This assumption is based on the fact that the value $\bar{h}_i^{-1}(Y^k)/\overline{\delta h^{-1}}$ is related to the maximum relative error of the neural network with respect to the input-output pairs that were used for training, while the goal of the training is to make the maximum error $\overline{\delta h^{-1}}$ as small as possible.

Starting from the closed loop equation (18) and using assumption 3 on $\bar{h}_i^{-1}(Y)$, we can write:

$$h_i^{-1}(Y) = g_{i,i}(\bar{h}_i^{-1}(\hat{Y}) - h_i^{-1}(Y) - \delta h_i^{-1}(Y)) \quad (24)$$

or, more in general:

$$Y = h(\bar{h}^{-1}(\hat{Y}) - (I+G)^{-1}\bar{h}^{-1}(\hat{Y}) - G(I+G)^{-1}\delta h^{-1}(Y)) \tag{25}$$

The right two terms are significantly smaller than $\bar{h}^{-1}(\hat{Y})$ since G is taken very large (assumption 8) and δh^{-1} is known to be very small. Therefore assumption 1 applies, and we can write equation (25) as:

$$Y = h(\bar{h}^{-1}(\hat{Y})) - [(I+G)^{-1}\bar{h}^{-1}(\hat{Y}) + G(I+G)^{-1}\delta h^{-1}(Y)]^T \frac{\partial h(\bar{h}^{-1}(\hat{Y}))}{\partial U} \tag{26}$$

Lemma 1 applies on $h(\bar{h}^{-1}(\hat{Y}))$, such that

$$Y = \hat{Y} + (\delta h^{-1}(\hat{Y}))^T \frac{\partial h(\bar{h}^{-1}(\hat{Y}))}{\partial U} - (\delta h^{-1}(Y))^T \frac{\partial h(\bar{h}^{-1}(\hat{Y}))}{\partial U} + [(I+G)^{-1}\delta h^{-1}(Y) - (I+G)^{-1}\bar{h}^{-1}(\hat{Y})]^T \frac{\partial h(\bar{h}^{-1}(\hat{Y}))}{\partial U} \tag{27}$$

Assumption 6 gives that $\delta h^{-1}(\hat{Y}) \equiv \delta h^{-1}(Y)$ and using corollary 1:

$$y_i = \hat{y}_i + \frac{1}{1+g_{i,i}}(\delta h_i^{-1}(Y) - \bar{h}_i^{-1}(\hat{Y})) \sum_{j=1,2} \frac{1}{\frac{\partial \bar{h}_j^{-1}(\hat{Y})}{\partial y_i}} \tag{28}$$

$$= \hat{y}_i + \Delta\hat{y}_i$$

The upper bound on $\delta h_i^{-1}(Y)$ is known (assumption 4) such that the upper bound on the final positioning error $\Delta\hat{y}_i$ can be calculated as:

$$\Delta\hat{y}_i \le \frac{1}{1+g_{i,i}}(\overline{\delta h^{-1}} + |\bar{h}_i^{-1}(\hat{Y})|) \sum_{j=1,2} \frac{1}{\left|\frac{\partial \bar{h}_j^{-1}(\hat{Y})}{\partial y_i}\right|} \tag{29}$$

The upper limit for the positioning error can be chosen by increasing the gain $g_{i,i}$. □

2.5. Remarks

A few practical conclusions for feedback control must be considered:

- $g_{i,i}$ can not always be taken arbitrarily high. In practise the gain is limited by the motor drivers, but also by the theoretical demand that $U \in \Omega$, such that $Y \in \Psi$ can be strictly guaranteed.
- In the case that h^{-1} is flattening (e.g. for nonlinear systems with higher order terms), feedback control showed to be noise sensitive. The reason lays mainly in the fact that output noise is hidden for the controller.
- If the assumption $\delta h^{-1}(\hat{Y}) \equiv \delta h^{-1}(Y)$ can not be made, the resulting output error will be limited to $\Delta Y < |h(\delta h^{-1}(\hat{Y}) - \delta h^{-1}(Y))|$. After convergence, a static error will remain that mainly depends on the quality of the neural network mapping.

3. Experimental Example

3.1. Description of the plant

A helicoidal spring is contracted using two computer controlled pulling cables K and L, that are connected at regular intervals inside the coil. The connections allow the cable to slide through in order to reduce the force between a cable and a section to radial components. By pulling the two cables, the spring bends over to a side and can be positioned at any point of a region in a 2-dimensional plane. This region is the region of interest Ψ of the plant.

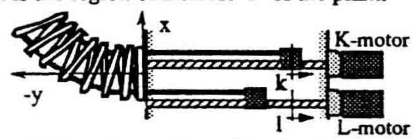

Figure 3. **Mounting of the cables inside the spring setup**

The motor speed values are updated by a Matlab routine that contains the controller in software. Due to the friction of the cables inside the spring, and low motor speeds, the spring can be considered as fully damped.

Starting from a random set of (Y_i, U_i) measurements it is possible to determine a (k_i, l_i) phase plane with measured motor positions. Ω is chosen as:

$$\Omega \subset \Re^2 : U_i \in \Omega ; U_i = (k_i, l_i) \tag{30}$$

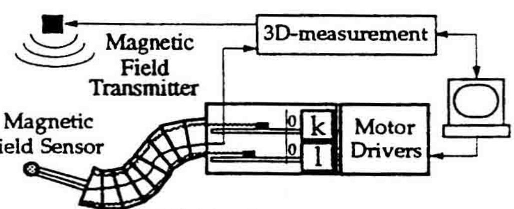

Figure 4. **Measurement setup**

The two motors cannot be placed at arbitrary positions. If one spindle would be placed at the far end while the other is fully pulled back, the spring would bend too far and the setup can get damaged. From the measurements it is possible to determine a (k, l) plane with static motor positions that will keep the end tip in a safe region of the measurement setup. This plane is used as a software bound for the motors in order to guarantee a safe operation of the spring.

3.2. A first approach using plain PI control

For reasons of comparison, the spring is controlled using