Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot

Jörg A. Walter and Klaus J. Schulten

Abstract— **We report on the implementation of two neural network algorithms for visuo-motor control of an industrial robot (Puma 562). The first algorithm uses a vector quantization technique, the "neural-gas" network, together with an error correction scheme based on a Widrow–Hoff-type learning rule. The second algorithm employs an extended self-organizing feature map algorithm. Based on visual information provided by two cameras, the robot learns to position its end effector without an external teacher. Within only 3000 training steps, the robot—camera system is capable of reducing the positioning error of the robot's end effector to approximately 0.1% of the linear dimension of the work space. By employing adaptive feedback the robot succeeds in compensating not only slow calibration drifts, but also sudden changes in its geometry. Hardware aspects of the robot—camera system are discussed.**

I. Introduction

THE adaptive capabilities of motion control of biological organisms are still highly superior to the capabilities of current robot systems. Therefore, various neural network models have been developed that apply biologically inspired control mechanisms [2], [6], [7], [11], [12], [15], [16] to robot control tasks. During the last two years it has been demonstrated by means of robot simulations that the neural network model [10], [12], [17], based on Kohonen's algorithm for self-organizing maps [4], [5], can be utilized for visuo-motor control. In the present paper we will report on an actual implementation of two newer versions of this algorithms for a system of two cameras and a PUMA 562, a robot arm widely used for industrial manufacturing.

The objective is to teach the system to position its end effector using information gained solely from the pair of cameras. Neither an external teacher nor any prior information about the geometry of the robot arm or the cameras will be employed.

The first algorithm uses the so-called "neural-gas" network, a new network algorithm introduced recently by Martinetz and Schulten [13] and inspired by Kohonen's feature map algorithm [4], [5]. In previous work, Martinetz *et al.* adopted it for the mentioned visuo-motor control problem and demonstrated its capabilities in several computer simulations. This work paved the way for implementing the method on an actual robot system and its evalution under real world conditions (some results have already been published in [19]). Since large parts of neural network research are devoted to the development of new algorithms, we also felt the need to compare the new method with the existing approaches. Therefore, we also implemented and evaluated the performance of the previous feature map algorithm and discuss the relative costs and merits of the two methods. The second algorithm is based on the extended self-organizing feature map algorithm [10], [17].

Random movement trials are used to train the network, consisting of between 100 to 400 neural units. Despite this small number of units, the robot accomplishes a positioning accuracy limited only by the resolution of the two cameras providing information about the spatial location of the target object. Furthermore, the robot system succeeds in quickly adapting to sudden, drastic changes in its environment. This adaptability is a major advantage over common commercial robot applications, which depend on precise calibration of all system components. Any distortion caused, e.g., by erosion or by a collision, requires a recalibration of the common robot system. Generally the resulting error will be repeated over and over again until the error is detected and an often time-consuming recalibration is carried out.

As demonstrated in this study both algorithms exhibit a good adaptivity on a fast time scale: a drastic change of the length of one of the arm segments is *compensated* using iterative visual feedback. If the change in length persists, the network *adapts* on a slower time scale, thereby gradually obviating the use of the feedback correction.

II. The Algorithm

The robot-camera system is shown schematically in Fig. 1. For each training step a target location is presented at a randomly chosen location within the workspace of the robot. Two cameras monitor the scene and a preprocessing unit identifies the two-dimensional image coordinates of the target location for each of the two camera images. The two pairs (u_1, u_2), (u_3, u_4) of image coordinates are combined into a four-dimensional vector $\boldsymbol{u}_{\text{target}} = (u_1, u_2, u_3, u_4)^T$. The set $\mathcal{U} \subseteq \mathbb{R}^4$ of all possible vectors $\boldsymbol{u}_{\text{target}}$ forms the input space of the network.

To be able to position its end effector correctly, the robot system has to know the transformation $\Theta(\boldsymbol{u}_{\text{target}})$ from $\boldsymbol{u}_{\text{target}}$ to the corresponding set of joint angles $\vec{\theta}$. The transformation depends (*i*) on the geometry of all the robot links, (*ii*) on the relative position of the robot with respect to the cameras, and (*iii*) on the camera's optical mapping properties. In our

Manuscript received October 3, 1991; revised April 6, 1992. This work was supported by the Beckman Institute for Advanced Science and Technology of the University of Illinois at Urbana-Champaign and by the National Science Foundation under Grant DIR-90-15561.

J. A. Walter is with the Department of Computer Science, University of Bielefeld, 4800 Bielefeld, Germany. email: walter@techfak.uni.bielefeld.de

K. J. Schulten is with the Beckman-Institute and Department of Physics, University of Illinois at Urbana-Champaign, Urbana, IL.

IEEE Log Number 9201253.

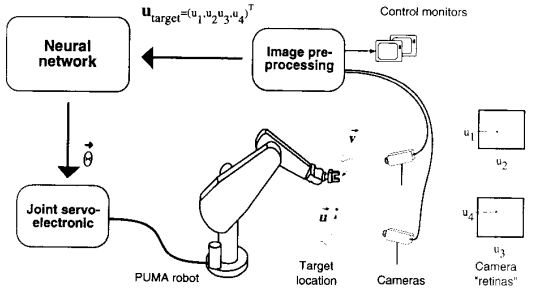


Fig. 1. Schematic representation of the robot vision system. Two cameras observe the target location ($\boldsymbol{u}_{\text{target}}$) and the location of the end effector ($\boldsymbol{v}$) of the robot arm in their "retinas." An image preprocessing unit extracts four-dimensional vectors ($\boldsymbol{u}_{\text{target}}$ shown) which describe three-dimensional workspace locations in camera coordinates. The PUMA robot arm is commanded by a set of joint angle commands $\vec{\theta}$ which constitute the output from the neural network.

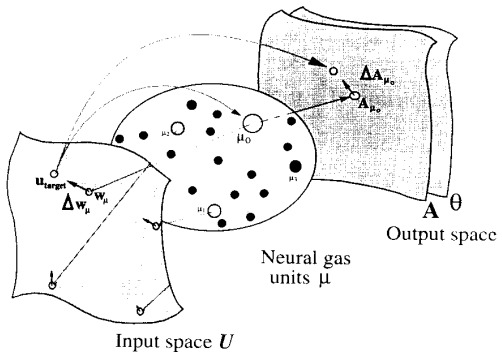


Fig. 2. Schematic representation of the "Neural-Gas" algorithm. *Left*: the 4-d input space $\mathcal{U}$ with the reference vectors $\boldsymbol{w}_\mu$ and the input vector $\boldsymbol{u}_{\text{target}}$ projected into it. Besides the reference vectors $\boldsymbol{w}$, each neuron μ (*middle layer*) stores two output quantities θ_μ and $\boldsymbol{A}_\mu$ (*right*) assigned to it. The arrows indicate the adaptation step, as explained in the text.

investigation we consider the nonredundant case of a robot arm with three degrees of freedom; the wrist is rigid. The same algorithm is likewise capable of controlling redundant degrees of freedom, as shown in computer simulations [10] . Besides the number of moveable joints, no further information about (*i*)–(*iii*) will be used. The aim is to learn the correct transformation $\Theta : \boldsymbol{u}_{\text{target}} \subseteq \mathcal{U} \mapsto \vec{\theta} \subseteq \mathbb{R}^3$ *without an external teacher*.

The principle idea behind our approach to learn this highly nonlinear transformation is to adaptively discretize the input space $\mathcal{U}$ into a set of N disjoint cells F_μ with $\mu \in \{1, 2, \ldots N\}$ and to approximate $\vec{\theta}$ within each cell by a linear mapping which is gradually refined as more trials movements are explored. This is illustrated in Fig. 2. To each cell F_μ we assign one neural unit, storing three different data items: the reference or weight vector $\boldsymbol{w}_\mu$, coding the center of the discretization cell, the output vector $\vec{\theta}_\mu$ and a matrix $\boldsymbol{A}_\mu$ that together specify a linear Taylor expansion of $\Theta(\boldsymbol{u}_{\text{target}})$ within the cell F_μ,

$$\vec{\theta}(\boldsymbol{u}_{\text{target}}) = \vec{\theta}_\mu + \boldsymbol{A}_\mu \cdot (\boldsymbol{u}_{\text{target}} - \boldsymbol{w}_\mu). \tag{1}$$

$\vec{\theta}_\mu$ is the zero order term and the 3×4-matrix $\boldsymbol{A}_\mu$ is a modified *Jacobian matrix*, which denotes the first order term. Both the choice of the discretization cells and the adaptive learning of their outputs $\vec{\theta}_\mu$ and $\boldsymbol{A}_\mu$ is achieved by means of the "neural-gas" network [13], [19] or, alternatively, by employing the extended self-organizing feature map algorithm as described below.

The learning procedure consists of a series of training steps. For each training step a target position in the workspace is presented and the vector $\boldsymbol{u}_{\text{target}}$ is extracted from the camera images. This is illustrated in Fig. 2 where the neural unit which has its reference vector $\boldsymbol{w}_\mu$ closest to the input vector $\boldsymbol{u}_{\text{target}}$ is labeled by μ_0. The selected unit μ_0 will be the main target of the subsequent training step and also will determine the output of the system $\vec{\theta}^{out}$ by virtue of (1) with $\mu = \mu_0$. Later we will modify this "winner takes all" scheme by involving neighboring neurons and arriving at a "winner takes most" algorithm.

A first coarse movement of the robot arm is carried out to attain the joint angles $\vec{\theta}_0^{\text{out}}$, to be specified later in (16). The subscript indicates the number of visual feedback steps (see below) used, $\vec{\theta}_0^{\text{out}}$ denotes the first feed forward result of the network. The resulting end effector location seen by the cameras is denoted by $\boldsymbol{v}_0$. With the Jacobian matrix any residual deviation from the desired target location ($\boldsymbol{u}_{\text{target}} - \boldsymbol{v}_0$) is transformed into a correctional fine movement $\vec{\theta}_1^{\text{out}}$ (18). To further reduce the remaining positioning error, this correcting fine movement can be repeated iteratively using the visual feedback information of the end effector locations $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots$. In order to gain the necessary information for the adaptation step, at least a single fine movement has to be carried out during the learning phase.

A. The Learning Rules

An important ingredient of the algorithm is to apply the learning steps not only to the "winner" but to a whole subset of neurons, thereby, increasing its rate of convergence. However, the adjustment has to be confined to a "neighborhood region" of the selected neuron, containing only neurons that have to learn similar output values and that therefore, can profitably participate in the same adjustment step. The essential difference between the two implemented algorithms is the definition of this "neighborhood." The "Extended Self-Organizing Feature Map" algorithm uses fixed neighborhood relations in contrast to the "Neural Gas" approach which builds these relations dynamical (for an alternative technique see [4]).

1) "Neural Gas" Algorithm In the "neural gas" algorithm the degree of neighborhood is given by the "rank of closeness" k of the neuron's reference vector to the input vector. For each new target location we assess the sequence $(\mu_0, \mu_1, \ldots, \mu_{N-1})$ of neural units by increasing order of their distance to the input vector $\boldsymbol{u}_{\text{target}}$, i.e.,

$$\|\boldsymbol{w}_{\mu_0} - \boldsymbol{u}_{\text{target}}\| < \|\boldsymbol{w}_{\mu_1} - \boldsymbol{u}_{\text{target}}\| < \cdots < \|\boldsymbol{w}_{\mu_{N-1}} - \boldsymbol{u}_{\text{target}}\|. \tag{2}$$

The learning step size for neuron μ_k is then chosen as a monotonously decreasing function of its *closeness rank order* k.

After completion of a trial all the neural units are adjusted according to

$$\boldsymbol{w}_{\mu_k} \leftarrow \boldsymbol{w}_{\mu_k} + \varepsilon \cdot g(k) \cdot (\boldsymbol{u}_{\text{target}} - \boldsymbol{w}_{\mu_k}) \tag{3}$$

$$\vec{\theta}_{\mu_k} \leftarrow \vec{\theta}_{\mu_k} + \varepsilon' \cdot g'(k) \cdot \Delta\vec{\theta}_{\mu_k} \tag{4}$$

$$\boldsymbol{A}_{\mu_k} \leftarrow \boldsymbol{A}_{\mu_k} + \varepsilon' \cdot g'(k) \cdot \Delta\boldsymbol{A}_{\mu_k}. \tag{5}$$

Here ε and $\varepsilon' \in [0, 1]$ scale the overall size of the adaptation step, and $g(k)$ and $g'(k)$ are functions of the closeness rank k (the primes denote parameters belonging to the output side). They have a maximum value of unity for the closest neural unit μ_0, i.e., $g(0) = 1$ and $g'(0) = 1$, and decrease to zero as k increases. A convenient choice is

$$g(k) = \exp\left(-\frac{k}{\lambda}\right); \qquad g'(k) = \exp\left(-\frac{k}{\lambda'}\right). \tag{6}$$

Here λ and λ' determine the size of the neighborhood region. Initially, their values are chosen fairly large for rapid, coarse adaptation of many neural units at each adaptation step, and they gradually decrease in the course of the learning phase in order to enable a fine-tuning of the system.

Fig. 2 illustrates the update step (3): the reference vectors are "pulled" toward the input vector $\boldsymbol{u}_{\text{target}}$. The differences of the arrow length reflect the modulation of the adaptation strength depending on the closeness rank k of the units μ_k (shown in different gray levels). This procedure generates a very homogeneous discretization of the relevant submanifold of the input space as explained in more detail in [13].

To complete the description of the adaptation equations (4) and (5), we still have to specify the quantities $\Delta\boldsymbol{A}_\mu$ and $\Delta\vec{\theta}_\mu$. $\Delta\boldsymbol{A}_\mu$ is chosen such that the step corresponds to a stochastic gradient descent for the quadratic cost function

$$E = \tfrac{1}{2}\left\langle (\Delta\vec{\theta}_{01}^{\text{out}} - \boldsymbol{A}_\mu \Delta\boldsymbol{v}_{01})^2 \right\rangle \tag{7}$$

$$\text{with} \Delta\boldsymbol{v}_{01} = \boldsymbol{v}_1 - \boldsymbol{v}_0 \qquad \text{and} \qquad \Delta\vec{\theta}_{01}^{\text{out}} = \vec{\theta}_1^{\text{out}} - \vec{\theta}_0^{\text{out}}, \tag{8}$$

and is given by an error correction rule of Widrow–Hoff type [21]

$$\Delta\boldsymbol{A}_\mu = \|\Delta\boldsymbol{v}_{01}\|^{-2}\left(\Delta\vec{\theta}_{01}^{\text{out}} - \boldsymbol{A}_\mu\Delta\boldsymbol{v}_{01}\right)\Delta\boldsymbol{v}_{01}^T \tag{9}$$

To obtain $\Delta\vec{\theta}_\mu$ for the correction of $\vec{\theta}_\mu$, we estimate a new $\vec{\theta}_\mu^*$ for the desired value of $\vec{\theta}_\mu$ and employ the locally valid relation $\vec{\theta}_0^{\text{out}} - \vec{\theta}_\mu^* = \boldsymbol{A}_\mu(\boldsymbol{v}_0 - \boldsymbol{w}_\mu)$. This yields

$$\Delta\vec{\theta}_\mu = \vec{\theta}_\mu^* - \vec{\theta}_\mu = \vec{\theta}_0^{\text{out}} - \vec{\theta}_\mu^{\text{old}} - \boldsymbol{A}_\mu(\boldsymbol{v}_0 - \boldsymbol{w}_\mu) \tag{10}$$

which completes our description of the learning rule for the "neural gas" algorithm.

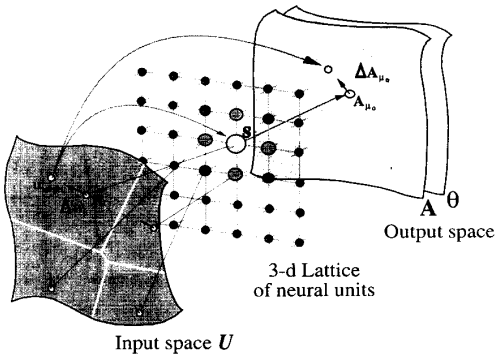


Fig. 3. Schematic representation of the extended feature map algorithm. In contrast to the previous figure the "neural gas" is substituted by the a lattice structure (*middle*). One planar slice of the cubic lattice of neural units is shown containing the selected neuron $\boldsymbol{s}$. The neighboring neural units are marked by different gray levels.

2) The Extended Self-Organizing Feature Map Algorithm (Fig. 3): The second algorithm is based on Kohonen's algorithm for self-organizing maps to adaptively discretize the input space [4], [5]. This algorithm was significantly extended in [18] by assigning matrix-valued output quantities to the neurons and thus taking advantage of its topology conserving properties in learning these output values. The network is now structured and consists of a Euclidean lattice of neural units of the same type as the neural gas units. The dimensionality of the lattice is chosen to fit the dimensionality of the work space. For the sake of distinction from the "neural gas" scheme we change the notation and label the neural units by their lattice position vectors $\boldsymbol{r}$ ($\boldsymbol{r} \in \mathbb{N}^3$) instead of μ. The selection of the "winning" neuron $\boldsymbol{s}$ with minimal distance between the reference vector and the input vector $\boldsymbol{u}_{\text{target}}$

$$\|\boldsymbol{w}_{\boldsymbol{s}} - \boldsymbol{u}_{\text{target}}\| = \min_{\forall\, \boldsymbol{r}} \|\boldsymbol{w}_{\boldsymbol{r}} - \boldsymbol{u}_{\text{target}}\| \tag{11}$$

is analogous to the selection of the neural unit μ_0 described above.

The learning rules are formally very similar to (3)–(5), i.e.,

$$\boldsymbol{w}_{\boldsymbol{r}} \leftarrow \boldsymbol{w}_{\boldsymbol{r}} + \varepsilon \cdot h_{\boldsymbol{r}\boldsymbol{s}} \cdot (\boldsymbol{u}_{\text{target}} - \boldsymbol{w}_{\boldsymbol{r}}) \tag{12}$$

$$\vec{\theta}_{\boldsymbol{r}} \leftarrow \vec{\theta}_{\boldsymbol{r}} + \varepsilon' \cdot h'_{\boldsymbol{r}\boldsymbol{s}} \cdot \Delta\vec{\theta}_{\boldsymbol{r}} \tag{13}$$

$$\boldsymbol{A}_{\boldsymbol{r}} \leftarrow \boldsymbol{A}_{\boldsymbol{r}} + \varepsilon' \cdot h'_{\boldsymbol{r}\boldsymbol{s}} \cdot \Delta\boldsymbol{A}_{\boldsymbol{r}}. \tag{14}$$

The crucial difference to the "neural gas" algorithm are the replacement of $g(k)$ and $g'(k)$ by the functions $h_{\boldsymbol{r}\boldsymbol{s}}$ and $h'_{\boldsymbol{r}\boldsymbol{s}}$. These functions are "bell-shaped" functions of the *lattice* distance $\|\boldsymbol{r} - \boldsymbol{s}\|$ from the selected neuron. Their effect is to specify a "neighborhood region" of the selected unit, as in the "neural gas" algorithm. However, the distance measure is different: instead of a distance *ranking* in the input space $\mathcal{U}$, the distance is now measured by means of the lattice metric.

A suitable choice for $h_{\boldsymbol{r}\boldsymbol{s}}$ is a Gaussian

$$h_{\boldsymbol{r}\boldsymbol{s}} = \exp\left(-\frac{\|\boldsymbol{r} - \boldsymbol{s}\|^2}{2\sigma^2}\right) \qquad h'_{\boldsymbol{r}\boldsymbol{s}} = \exp\left(-\frac{\|\boldsymbol{r} - \boldsymbol{s}\|^2}{2\sigma'^2}\right). \tag{15}$$

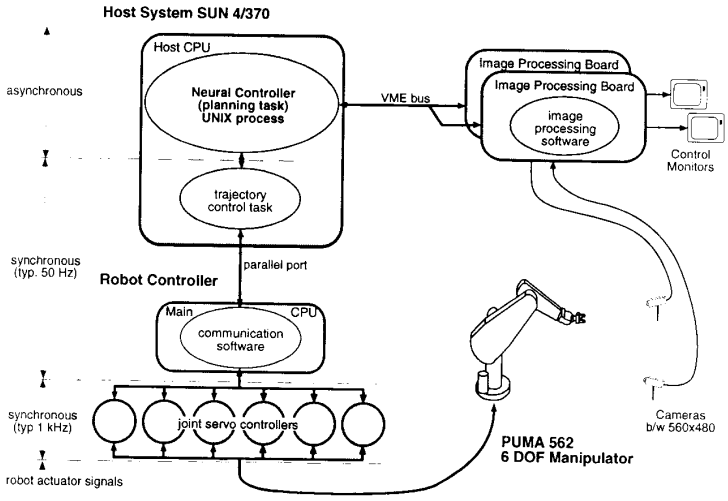


Fig. 4. The main hardware components of the robot system.

Parameters σ and σ' play similar roles as λ and λ' in the "neural gas" scheme and determine the size of the neighborhood region in a learning step.

B. Averaged Output

The transformation $\vec{\theta}(\boldsymbol{u}_{\text{target}})$ is locally approximated by (1). For the positioning movement, however, not only the linear approximation associated with the unit μ_0 "closest" to $\boldsymbol{u}_{\text{target}}$ determines the joint angles, but the output of a whole subset of neural units with their vectors $\boldsymbol{w}_\mu$ close to $\boldsymbol{u}_{\text{target}}$ is taken into account. This is achieved by averaging (1) over all neural units μ_k, weighted by the function $g'(k)$ (6) or weighted by a more general but similar function $g^{\text{mix}}(k)$. Similar to (3), the neural unit μ_0 with its $\boldsymbol{w}_{\mu_0}$ closest to $\boldsymbol{u}_{\text{target}}$ contributes the most, for the unit μ_1 with its $\boldsymbol{w}_{\mu_1}$ second closest to $\boldsymbol{u}_{\text{target}}$ the contribution is second largest, and so forth.

The joint angles which the network produces to reach the target are then given by

$$\vec{\theta}_0^{\text{out}} = s^{-1} \cdot \sum_k g^{\text{mix}}(k)\left(\vec{\theta}_{\mu_k} + \boldsymbol{A}_{\mu_k}(\boldsymbol{u}_{\text{target}} - \boldsymbol{w}_{\mu_k})\right) \quad (16)$$

where

$$s = \sum_k g^{\text{mix}}(k) \qquad \text{and} \qquad g^{\text{mix}}(k) = \exp\left(-\frac{k}{\lambda^{\text{mix}}}\right). \quad (17)$$

Similar to the coarse movement defined through (16) the corrective fine movement upon the nth visual feedback $\boldsymbol{v}_n$ leads to the weighted averaged network output

$$\vec{\theta}_n^{\text{out}} = \vec{\theta}_{n-1}^{\text{out}} + s^{-1} \cdot \sum_k g^{\text{mix}}(k)\boldsymbol{A}_{\mu_k} \cdot (\boldsymbol{u}_{\text{target}} - \boldsymbol{v}_n). \quad (18)$$

The extended self-organizing feature map algorithm uses an analogous output averaging scheme, i.e., $g(k)^{\text{mix}}$ is substituted by $h_{\boldsymbol{r}s}{}^{\text{mix}} = \exp(-(\|\boldsymbol{r} - \boldsymbol{s}\|/\sigma^{\text{mix}})^2/2)$ and $\boldsymbol{r}$ replaces μ_k, and k.

III. The Set-Up

Complex control algorithms for a robot require the capability to quickly process and respond to high bandwidth sensory input. The design of a robot vision system which provides a testbed for neural algorithms has to overcome the limitations of today's commercially available robot controllers. These limitations are a lack of computational power, lack of expandability or compatibility to other systems, and little transparency of their programming languages or operating systems. In our implementation we chose to replace large parts of the original Puma controller software and employed a Unix workstation to *directly* control the robot in real-time via a high speed communication link.

A. Components

The main components of our system are illustrated in Fig. 4. The Puma 562 is a 6-degree of freedom manipulator which is connected to the Unimation Controller (Mark III, *bottom left*) which itself contains several controllers. The separate servo controllers—one for each revolute joint—are commanded by a main controller LSI-11/73 CPU.

In industrial applications the robot is programmed in an interpreted robot language VAL II. We chose to replace the VAL II software, since the main controller LSI-11 is not capable of handling high bandwidth sensory input itself, e.g., from a video camera, and VAL II does not support flexible control by an auxiliary computer. To achieve a tight real-time control directly by the Unix workstation we installed the software package RCCL/RCI (*R*obot *C*ontrol *C* *L*ibrary and *R*eal-time *C*ontrol *I*nterface) [8]. This package allows the user to issue robot motion requests from a high level control program ("planning task" which is written and executed as an ordinary C program) to the trajectory control level ("control task") via shared memory communication. The control task is executed periodically at a high priority (kernel mode) and is responsible for reading feedback data, generating interme-

diate joint setpoints, and carrying out a "watchdog" function (collision avoidance). During each control cycle (typically 20 ms) a command package is sent to the robot controller via the parallel port. The receiving main controller LSI-11 CPU is reprogrammed to dispatch commands to the joint servos, collect feedback data from them, and perform elementary safety checks. At power-up time the reprogramming software is downloaded and started by the host computer through a serial line, emulating the controller console. The software then resides at the controller and can be addressed through the parallel port.

The Unix workstation is a VME—based SUN Sparcsystem 4/370, which hosts some interfacing hardware and two image processing boards (ICS-400, Androx Inc., MA), each based on four digital signal processors. These boards provide the computing power for fast image preprocessing and feature extraction. The sensory input comes from two monochrome CCD cameras (560×480 resolution), oriented toward the robot's workspace with a disparity angle of about 50°.

B. Image Data Extraction

To keep the problem of image segmentation simple during the initial stage of our research, we mounted a miniature lamp on the gripper hand (lamp power controlled by software). This object is visually well defined and can be easily discriminated against most backgrounds. To identify the location of the lamp in one of the two camera images, the pixel set $\Pi_{\max}$ of the currently maximally bright pixels in the video frame is determined. The center of the minimal rectangle Π_{rect} enclosing $\Pi_{\max}$ is taken to approximate the exact target location in image coordinates, which implies a resulting discretization of u_i in 0.5 pixel intervals. A subsequent consistency test assures identification of the lamp rather than identification of a spurious reflex. It includes checking the geometrical size of $\Pi_{\max}$ (which should be of the order of 5 pixels), minimal covering ratio of Π_{rect} by $\Pi_{\max}$. In dubious cases, a minimal response $\beta_{\text{on}}^{\max} - \beta_{\text{off}}^{\max}$ to turning off the lamp is verified.

C. Training Procedure

During the learning phase several hundred randomly chosen targets within the workspace must be presented to the robot. This can be achieved very conveniently by operating the robot in what we call a "*split brain*" fashion: the controlling program alternates between the following two modes without passing any information except the camera input.

Set target mode: In this mode, the robot arm is used to indicate the target location to the camera system. This position is randomly chosen with a uniform distribution in joint angle configuration space. The cameras view the resulting position of the end effector $\boldsymbol{u}_{\text{target}}$ and the arm returns to its previous configuration or any other configuration.

Retrieve position from camera information: The neural network algorithm outputs the joint angles $\vec{\theta}_0^{\text{out}}(\boldsymbol{u}_{\text{target}})$ for the coarse movement, followed by the fine movement $\vec{\theta}_1^{\text{out}}(\boldsymbol{v}_0)$ after processing visual feedback information $\boldsymbol{v}_1$. For a maximum positioning accuracy the feedback control loop might be repeated at this point (see Section IV-A-3)). During the learning phase the adaptation steps (3)–(5) and (12)–(14) are performed.

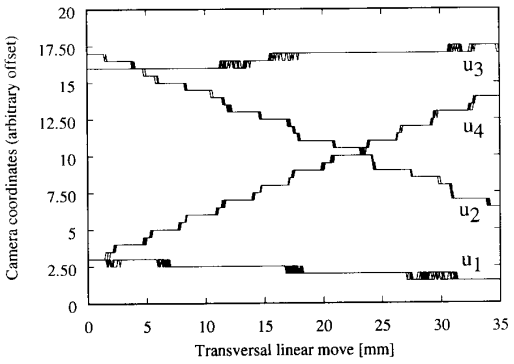


Fig. 5. Typical result of a straight test move of the end effector (here in a transversal horizontal direction). Plotted are the extracted camera coordinates (u_1, u_2, u_3, u_4) versus the linear displacement (with arbitrary offsets). The plateaus are caused by the discretization in 0.5 pixel steps and the flanks are broadened by repeating the path interval five times (round trip). The comparison of the flank width and the plateau width, proves that the repetition accuracy of the robot arm is higher than the resolution of the image preprocessing.

The workspace consists of a sphere segment, oriented toward the cameras and enclose a volume of approximately 1 m^3 while excluding a safety zone above the table and behind the robot base. In the neighborhood of the "elbow singularity," which denotes the manipulator configuration with a straight arm, a small arm straightening translates to a very large change in the elbow joint angle. The Jacobian matrices $\boldsymbol{A}$ are ill-conditioned in these remote workspace areas, therefore, their values cannot converge to a meaningful value and positioning ability is significantly reduced. To separate these effects, the network was trained and tested using a sphere segment with a radius that was about 10 cm smaller than the possible maximum.

D. Performance Measurement

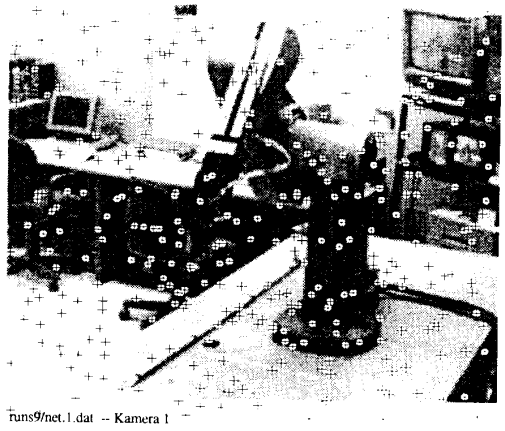
The performance of the network is monitored by computing the average Euclidean distance between target and end effector position. Due to the "split brain" procedure this can be done with sufficient accuracy by evaluating the well-known geometry of the robot (e.g., [1]) and differences of joint angles, read from the calibrated built-in encoders.

Fig. 5 shows the experimental result of an accuracy test (for details see figure caption). The test demonstrates that the measured repetition accuracy of the robot arm is about one order of magnitude better than the resolution of the image preprocessing and, therefore, the method for performance monitoring is sufficient for our purpose. The particular measurement method captures gear and encoder imprecisions as well as pixel jitter.

IV. Experimental Results

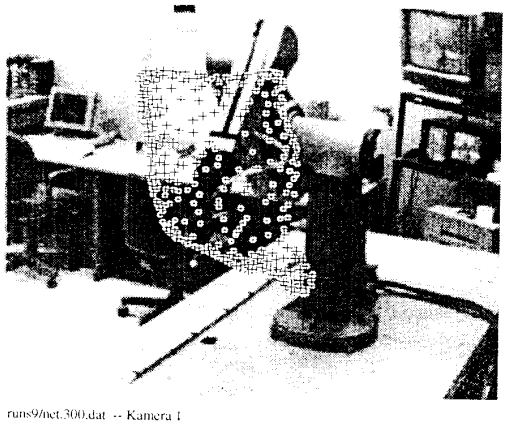
A. The "Neural-Gas" Algorithm

1) Parameters and Initialization: In this section we present

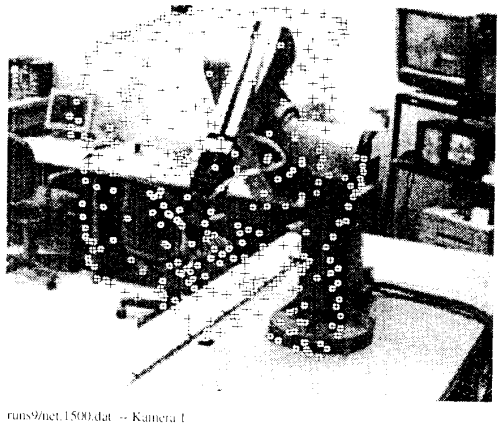


(a)

(b)



(c)



(d)

Fig. 6. (a)–(d) The development of the network seen as a projection of the reference vectors $\boldsymbol{w}$ onto the image planes of the cameras. Shown are the stages after t learning steps, with (a) $t = 1$, (b) $t = 100$, (c) $t = 300$, and (d) $t = 1500$ *(con't.)*

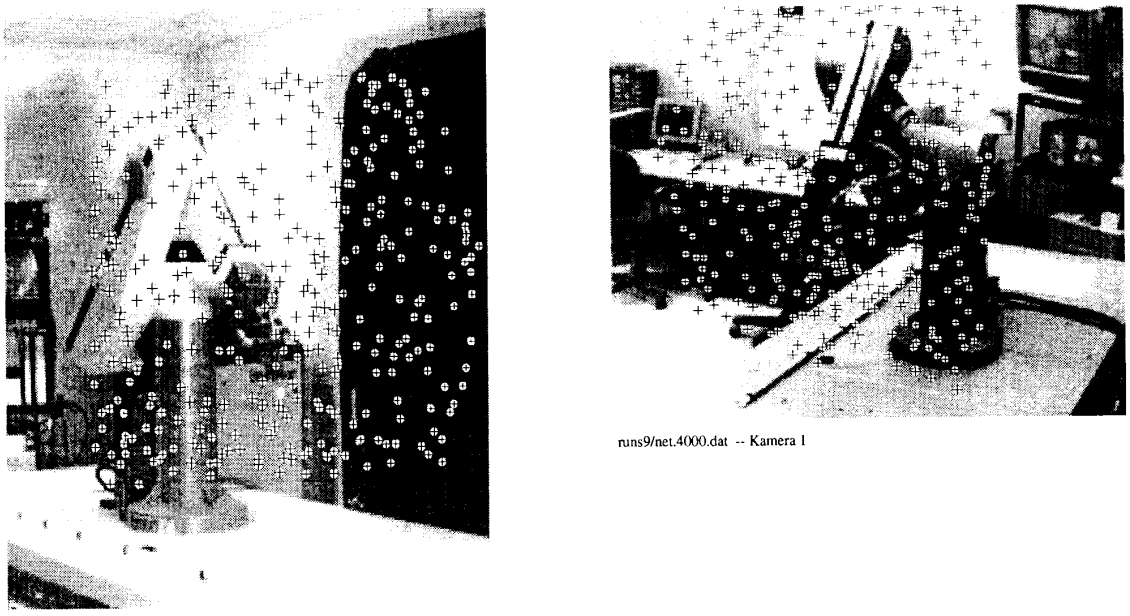
the experimental results obtained by applying a "neural gas" network with $N = 300$ units. The parameters $\varepsilon, \varepsilon'$ and the widths λ, λ', λ^{mix} all had the same time dependence $p(t) = p_i(p_f/p_i)^{t/t_{\max}}$ with t as the number of already performed learning steps and $t_{\max} = 4000$. The initial and final values were chosen as $\varepsilon_i = 0.3, \varepsilon_f = 0.05, \varepsilon'_i = \varepsilon'_f = 0.9, \lambda_i = 150, \lambda'_i = \lambda_i^{\mathrm{mix}} = 50$ and $\lambda_f = \lambda'_f = \lambda_f^{\mathrm{mix}} = 1$. The Jacobians $\boldsymbol{A}_\mu$ were initialized by assigning a random value from the interval $[\pm 2.7'/\mathrm{pixel}]$ to each element. A similar procedure was used for the initialization of the joint angles $\vec{\theta}_\mu$.

2) Evolution of the Reference Vectors $\boldsymbol{w}$ Fig. 6(a)–(f) presents a sequence of learning stages after $t = 0$, $t = 100$, $t = 300$, $t = 1500$, and $t_{\max} = 4000$ positioning trials. The reference vector of each neural unit is visualized by projecting $\boldsymbol{w}$ onto the image plane of each camera. Initially, the vectors $\boldsymbol{w}$ are distributed randomly in the image of the cameras (left upper picture). After about 2000 learning steps the initial distribution has retracted from the four-dimensional input space (as initialized) to the relevant three-dimensional subspace corresponding to the actual workspace. Finally, a regular distribution of the neural units emerged, reflecting the economical and demand-driven allocation of computational resources effected by the neural network.

3) Positioning Performance and Adaptation Capabilities: Fig. 7 presents the averaged positioning accuracy versus the number of already experienced learning steps. The upper curve shows the average[1] Euclidean deviation from the desired target location after the initial coarse movement (open loop system) and the lower curves represent this positioning accuracy after using iteratively 1, 2, 3, 4, and 5 visual feedback cycles[2] for correctional movements. Initially the position accuracy increases rapidly and reaches an average positioning error of 1.3 mm within the first 3000 training steps. A comparison of the curves illustrates the substantial improvement of using visual feedback.

[1] Moving average: $\bar{F}(t) = \alpha F(t) + (1 - \alpha)\bar{F}(t - 1)$ with $\alpha = 0.03$.

[2] A single movement (18) per learning step is the minimum to obtain necessary feedback for learning; the corresponding curve is drawn bold.



(e) (f)

Fig. 6.*(con't.)* Stage after t learning steps with (e) and (f) $t = t_{\max} = 4000$ (both cameras).

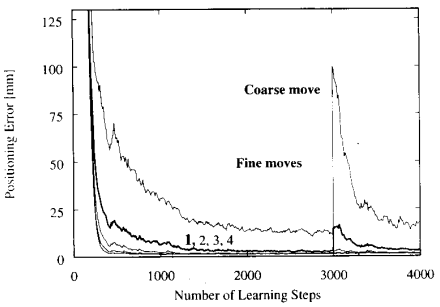


Fig. 7. The positioning capabilities of the "neural gas" network over the course of learning. Shown is the averaged error after the coarse movement (open loop: *upper curve*), and after 1, 2,... 5 iterative feedback loops ("fine moves"). After 3000 learning steps the last arm segment was suddenly elongated by 100 mm ($\simeq$ 10% of the linear dimensions of the workspace).

A very important advantage of self-learning algorithms is their ability to adapt to different and changing environments. To demonstrate the adaptability of the network, we interrupted the learning procedure after 3000 training steps and extended the last arm segment by 100 mm. The right side of Fig. 7 displays how the algorithm responded. After this drastic change of the robot's geometry only 300 further iterations were necessary to re-adapt the network for regaining the robot's previous positioning accuracy. Employing visual feedback the network is able to adapt while compensating immediately for the distortion.

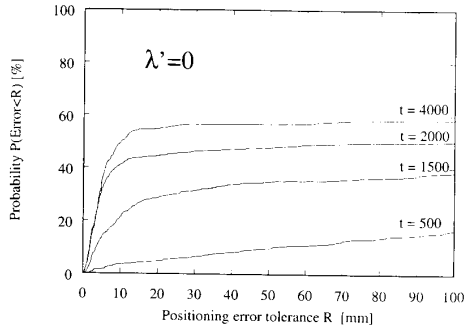
4) Influence of Collective Learning: Next, we turn to the following question: how much does the collective adaptation of the output of neighboring neural units influence the learning capabilities of the system? To answer this question we compare two learning runs, one regular run and one with a nonshared output learning scheme ($\lambda' \to 0$).

When positioning skills are poor—as they are in the early learning phase, and throughout the entire latter run—the network might request motor commands which would drive the robot arm into prohibited areas, like the table zone. These requests are rejected. Consequently, these deficient trials cannot be recorded by averaging the absolute positioning errors as shown in Fig 7. To obtain a performance measure that also accounts for these faulty trials, we now consider the probability $P_t(R)$ to stay at a time t within a given error margin R. $P_t(R)$ can be written as

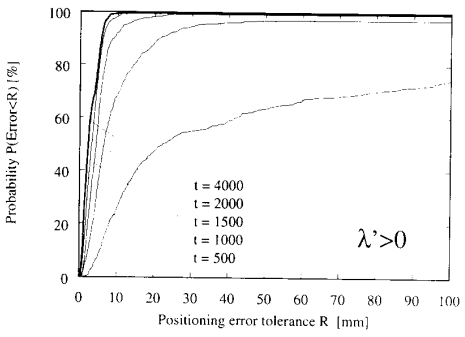
$$P_t(R) = \int_0^R p_t(r)dr \tag{19}$$

when r is the Euclidean positioning error. Here $p_t(r)dr$ is the probability that for a trial at time t the error is found in the interval $[r, r + dr]$.

The upper graph of Fig. 8 shows $P_t(R)$ for a system of units learning independently ($\lambda' = 0$ in (6). As can be seen the positioning skill, shown at $t = 500$, $t = 1000$, $t = 2000$, and $t = t_{\max} = 4000$, increases rather slowly. The resulting control is still insufficient, each third effort is erroneous. The lower graph of Fig. 8 shows $P_t(R)$ for a system of units learning collectively. The results show that after only a few hundred trials the algorithm already performs better than the independently learning system ever achieves to perform. The results for $t = 500$, $t = 1000$, $t = 1500$, $t = 2000$, and $t_{\max} = 4000$ demonstrate that the positioning skill develops rapidly in the beginning and achieves asymptotically an accurate performance.



(a)



(b)

Fig. 8. (a) Positioning ability with and (b) without collective learning of the output values. Plotted is the probability $P_t(R)$ to keep a given error tolerance R at different learning stages t after one feedback loop.

5) Scaling Behavior: Fig. 9 indicates the scaling behavior of the final averaged positioning accuracy after 4000 learning steps versus the number N of employed "neural gas" units. λ and λ' are scaled in linearly with N. The upper curve depicts the coarse move's outcome, the lower curves show the result for an increasing number of visual feedback loops. As one can easily see, the more neurons one employs, the fewer feedback loops are needed to achieve the same precision.

B. The Extended Self-Organizing Feature Map Algorithm

1) Parameters and Initialization: In this section we will compare results from an experiment using the extended self-organizing feature map algorithm with a 7×7×7 lattice of neural units.

The initialization was similar to the method described in Section A and the parameters $\varepsilon, \varepsilon'$ and the widths σ, σ^{mix} also had the same exponentially decaying time dependence. The values were chosen as $\varepsilon_i = 1, \varepsilon_f = 0.01, \varepsilon'_i = \varepsilon'_f = 0.9$, $\sigma_i = 2.5$, $\sigma_f = 0.01$, $\sigma_i^{\text{mix}} = 1.$ and $\sigma_f^{\text{mix}} = 0.1$. The Jacobians $\boldsymbol{A}_\mu$ and the joint angles $\vec{\theta}_\mu$ are randomly initialized similar to the "neural gas" algorithm. In contrast to the latter algorithm the network has first to "untangle" itself for the purpose of creating a topological ordering of the network. This ordering is essential to ascertain a meaningful participation of neighbored neurons in the learning phase. To keep the initial ordering time to a minimum the initial learning parameters $\varepsilon_i, \varepsilon'_i, \sigma_i, \sigma'_i$ must be relatively large. For further quick convergence the neurons have to decouple rapidly (σ, σ' decreasing) and must be adjusted by finer and finer learning steps (ε decreasing). In order to achieve a rapid convergence and keep good adaptive capabilities of the algorithm in the later learning phase, we chose to first decrease σ' exponentially from $\sigma'_i = 2.5$ to 0.5 at $t = 2000$ learning steps and keep it constant from then on.

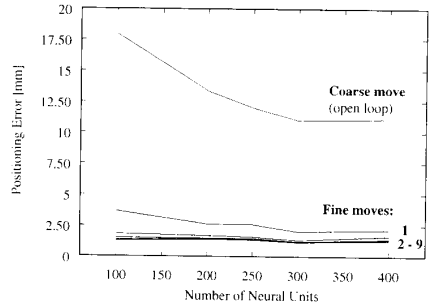


Fig. 9. The final positioning error as a function of the number of neural units employed in the learning scheme. Shown are the averaged positioning error without visual feedback ("coarse move") and after 1, 2,... 9 feedback loops using visual feedback information.

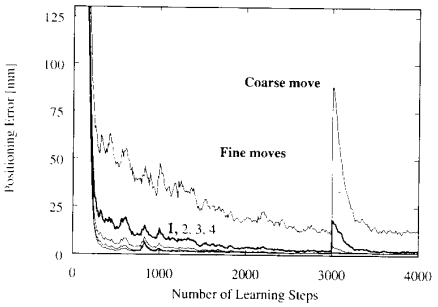


Fig. 10. The positioning error of the extended Kohonen network over the course of learning. Similar to the previous figure, the averaged error is shown after the coarse movement (open loop: *upper curve*) and after 1, 2,... 5 iterative feedback loops ("fine moves"). After 3000 learning steps the last arm segment was suddenly elongated by 100 mm ($\simeq 10$.

2) Positioning Performance and Adaptation Capabilities: Fig. 10 shows the records of the same experiment as constructed for the "neural gas" scheme presented in Fig. 7. The comparison of the two graphs demonstrates the qualitative and quantitative equivalence of these two algorithms. The final positioning skills are equally accurate (1.3 mm) and a sudden change of the robot's arm length leads to the same adaptive response as in the case of the "neural gas" algorithm.

V. Discussion and Conclusions

We investigated two visuo-motor control algorithms for the positioning task of an industrial robot system, the "neural gas" and the extended self-organizing feature map algorithm. Both

algorithms followed the concept of adaptive partitioning of the input space and storage of local linear maps (LLM). The major difference between them is the definition of "neighborhoods," which enable a successful participation of neural units in learning steps. For the given control problem both algorithms performed similarly. The main benefit of the "neural gas" algorithm becomes obvious when the topology of the submanifold in the input space is unknown or inhomogenous as shown in [13] for a constructed partly one-, two- and three-dimensional input space and in [9] for the fractal submanifold of a chaotic time series attractor.

The extended self-organizing feature map algorithm is slightly more dependent on proper tuning of the learning parameters than the "neural gas" algorithm. As mentioned above, in the worst case of a random initialization of the reference vectors $\boldsymbol{w_r}$ the network needs some time to "untagle" and "unfold" in the input space. The amount of time depends strongly on the learning parameters $\varepsilon(t)$ and $\sigma(t)$.

Compared to the predecessor versions [10], [17], of the introduced learning algorithms a significantly increased learning rate was accomplished. The reduction of the required number of trial movements is due primarily to an "individualization" of the neural units in the learning scheme. Former versions [11], [17] have applied the update rules (9) and (10) only to the "winning" unit μ_0 (or $\boldsymbol{s}$), thus, implicitly defining new estimates $\vec{\theta}^*$ and $\boldsymbol{A}^*$ for the desired values of the linear map (1) by

$$\Delta \boldsymbol{A}_{\mu_0} =: \boldsymbol{A}^* - \boldsymbol{A}_{\mu_0} \qquad \Delta \vec{\theta}_{\mu_0} =: \vec{\theta}^* - \vec{\theta}_{\mu_0}. \tag{20}$$

Subsequently, $\vec{\theta}^*$ and $\boldsymbol{A}^*$ have served also all other neural units μ as the current, universal goal value toward which they have been "pulled"

$$\Delta \boldsymbol{A}_{\mu} := \boldsymbol{A}^* - \boldsymbol{A}_{\mu} \quad , \qquad \Delta \vec{\theta}_{\mu} := \vec{\theta}^* - \vec{\theta}_{\mu} \ , \tag{21}$$

further modulated by the neighborhood function $\varepsilon' g'(\mu)$ (or $\varepsilon' h'_{\boldsymbol{r}s}$) as described in (4)–(5) and (13)–(14).

This form of *tight cooperation* is now replaced by the more *collective* and *parallel* learning scheme (*all* units apply (9) and (10)), enabling the units to learn independently from the trainings examples. Particularly in the initial training phase, large neighborhood size parameters λ and λ' (or σ, σ') are desirable, coupling also units which have to learn rather different output values. The independent learning significantly reduces the time consuming effect of mutual deflection from already well learned values for the Jacobian matrices $\boldsymbol{A}_{\mu}$ (or $\boldsymbol{A}_r$). Subsequently it generates a quicker convergence of the $\vec{\theta}_{\mu}$ (or $\vec{\theta}_{\boldsymbol{r}}$) output values (10).

This *collective* learning scheme was first used for nonlinear time series prediction with self-organizing maps [20] and adopted by Martinetz for the "neural gas" algorithm for the visuo-motor control problem [14], [19] as presented in this paper.

Both learning algorithms described above achieve a final positioning accuracy of 1.3 mm, or 0.1% of the linear dimension of the workspace of the robot arm. Furthermore, the system succeeded to rapidly adapt to drastically changing situations. These algorithms achieve a precision that is higher by one order of magnitude than earlier neural network implementations, like Kuperstein's system with 4%–6% average deviation [6], [7].

The accuracy is currently limited by the image processing resolution and not by the control algorithm. There are several fairly simple possibilities to enhance the performance: the use of additional cameras for accuracy sensitive parts of the work space or employing cameras with better resolution. Another possibility would be to employ the gray value information in the gray image and resolve $\boldsymbol{u}_{\text{target}}$ and $\boldsymbol{v}$ to subpixel precision.

We conclude that the neural-gas algorithm with collective neighborhood learning provides an efficient, robust and accurate learning scheme for real world robot control.

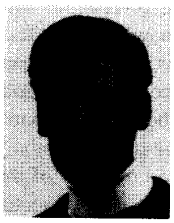
ACKNOWLEDGMENT

The work represented in this paper owes much to T. Martinetz, who paved the way for the present implementation by carrying out valuable computer simulations and providing code for the "neural gas" algorithm. Thanks also to H. Ritter for proofreading the manuscript. The authors would like to thank N. Ahuja for providing the Puma robot for this project.

REFERENCES

[1] J. Craig. *Introduction to Robotics.* Addison-Wesley, 1986.
[2] R. Eckmiller, J. Beckmann, H. Werntges, and M. Lades, "Neural kinematics net for a redundant robot arm," in *IJCNN, Washington DC*, vol. 2, pp. 333–340, 1989.
[3] J. A. Kangas, T. Kohonen, and J. T. Laakson, "Variants of self-organizing maps," *IEEE Trans. Neural Networks*, vol. 1, pp. 93–99, 1990.
[4] T. Kohonen, "Self-organized formation of topographically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59–69, 1982.
[5] T. Kohonen, *Self-organization and Associative Memory, Springer Series in Information Science.* Berlin, Heidelberg, New York: Springer, 1984, vol. 8.
[6] M. Kuperstein, "Adaptive visual-motor coordination in multijoint robots using parallel achitecture," in *IEEE Int. Automat. Robotics* , Raleigh, NC, pp. 1596–1602, 1987.
[7] M. Kuperstein, "Neural model of adaptive hand-eye coordination for single postures," *Science*, vol. 239, pp. 1301–1311, 1988.
[8] J. Lloyd, M. Parker, and R. McClain., "Extending the RCCL programming environment to multiple robots and processors," in *IEEE Conf. Robotics and Automat.*, Philadelphia, PA, pp. 465–469, Apr. 1988.
[9] T. Martinetz, S. Berkovich, J. Walter, and K. Schulten, "'Neural gas' network for vector quantization and its application to time series prediction," in preparation, 1992.
[10] T. Martinetz, H. Ritter, and K. Schulten, "Learning of visuomotor coordination of a robot arm with redundant degrees of freedom," in *Proc. ICNC-90, Düsseldorf*, North Holland, Amsterdam, Mar. 1990, pp. 431–434.
[11] T. Martinetz, H. Ritter, and K. Schulten, "Three-dimensional neural net for learning visuomotor coordination of a robot arm," *IEEE Trans. Neural Networks*, vol. 1, pp. 131–136, Mar. 1990.
[12] T. Martinetz and K. Schulten, "Hierarchical neural net for learning control of a robot's arm and gripper," in *Proc. IJCNN-90, San Diego*, June 1990, vol. 3, pp. 747–752.
[13] T. Martinetz and K. Schulten, "A "neural gas" network learns topologies," in *Proc. Int. Conf. Artificial Neural Networks, Espoo Finland*, Kohonen *et al.*, Eds. New York: Elsevier, June 24–28, 1991, vol. 1, pp. 397–407.
[14] T. Martinetz, J. Walter, and K. Schulten, "A neural network with Hebbian-like adaptiation rules learning visuomotor coordination of a robot arm," submitted for publication, 1991.
[15] B. W. Mel, "A robot that learns by doing," presented at the *AIP* Neural Informat. Processing Syst. Conf., Denver, CO, 1987.
[16] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki. Feedback-error-learning neural network for trajectory control of a robotic manipulator, *Neural Networks*, vol. 1, pp. 251–265, 1988.
[17] H. Ritter, T. Martinetz, and K. Schulten, "Topology-conserving maps for learning visuo-motor-coordination," *Neural Networks.*, vol. 2, pp. 159–168, 1989.

[18] H. Ritter and K. Schulten, "Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements," in *Neural Computers*, R. Eckmiller and von der Malsburg, Eds. Heidelberg, Germany: Springer, pp. 393–406, 1987.
[19] J. Walter, T. Martinetz, and K. Schulten, "Industrial robot learns visuomoto coordination by means of "neural-gas" network," in *Proc. Int. Conf. Artificial Neural Networks*,, Espoo, Finland, June 24–28, 1991, Kohonen *et al.*, Eds. New York: Elsevier, vol. 1, pp. 357–364.
[20] J. Walter, H. Ritter, and K. Schulten, "Non-linear prediction with self-organizing maps," in *Int. Joint Conf. Neural Networks*,, San Diego, CA, June 1990, vol. 1, pp. 587–592.
[21] B. Widrow and M.E. Hoff, "Adaptive switching circuits," *WESCON Conv. Record*, vol. 4, pp. 96–104, 1960.



Jörg A. Walter was born in 1963. He studied physics and engineering at the Technical University of Munich, Germany and the University of Illinois. He received the Diplom degree in physics from the Technical University of Munich.

In 1989 he spent three months at the International Institute of Applied System Analysis in Laxenburg, Austria and moved afterwards to the University of Illinois at Urbana-Champaign and the newly established Beckman Institute. His main research focuses on the design and implementation of neural network architectures for robot vision and movement control. He has also carried out research on the long term consequences of the Greenhouse effect and the time series prediction with neural networks. In 1991 he has moved to the University of Bielefeld, Germany where he is carrying out research in the Department of Information Science.

Klaus J. Schulten was born in 1947. He received the Diplom degree in physics from the University of Münster, Germany, in 1969, and the Ph.D. degree in chemical physics from Harvard University in 1974.

In 1974 he joined the Max-Plank-Institute for Biophysical Chemistry in Göttingen and in 1980 became Professor of Theoretical Physics at the Technical University of Munich. In 1988 he moved to the University of Illinois at Urbana-Champaign where he is Professor of Physics, a member of the new Beckman Institute and Director of the National Institutes of Health Resource for Concurrent Biological Computing. His research areas are theoretical physics, theoretical biology and computational biology, in particular, computational neural science. In the latter area he studies currently the problems of biological motion control and biological vision.