Proc. ICANN-91, Helsinki Finland, June 1991; Elsevier Amsterdam

Industrial Robot Learns Visuo-motor Coordination by Means of "Neural-Gas" Network

Jörg A. Walter, Thomas M. Martinetz, and Klaus J. Schulten

Beckman-Institute and Department of Physics
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
Email: walter@lisboa.ks.uiuc.edu

Abstract — **We implemented a neural network algorithm for visuo-motor control of an industrial robot (Puma 562). The algorithm uses a new vector quantization technique, the "neural-gas" network, together with an error correction scheme based on a Widrow-Hoff-type learning rule. 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 "closed" robot–camera system is capable of reducing the positioning error of the robot's end effector to approximately 0.2 percent of the linear dimension of the work space. By employing immediate feedback the robot succeeds in compensating not only slow calibration erosion, but also sudden changes in its geometry. Some hardware aspects of the robot–camera system are also discussed.**

1. Introduction

The adaptive capabilities of motion control of biological organisms are very much superior to capabilities of current robot systems. Various neural network models have been developed that apply biologically inspired control mechanisms [1, 4, 5, 7, 8, 10] to robot control tasks. During the last two years it has been demonstrated by means of robot simulations that the neural network model described in [7, 8], that is based on Kohonen's algorithm for self-organizing maps [3], can be utilized for visuo-motor control. In the present paper we will report on an actual implementation of a new version of this network for the system of two cameras and a PUMA 562, a robot arm widely used for industrial manufacturing. The algorithm now uses the socalled "neural-gas" network, a network suggested recently [9], which has been inspired by Kohonen's feature map algorithm [2, 3]. The objective is to teach the system to position its end effector using solely information gained 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.

To train the robot we employed a network of 300 neural units. Despite this small number of units, the robot accomplished 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 succeeded in quickly adapting to sudden, drastic changes in its environment. A change of the length of one of the arm segments was partially compensated immediately. Subsequently the neural network slowly regained its previous accuracy. This adaptablitiy is a major advantage compared to common commercial robot applications which depend on precise calibration of all system components: any distortion in the geometric arrangement of their mechanical components requires a recalibration of the robot system.

2. The Algorithm

The robot-camera system is presented schematically in Figure 1 (bottom right). For each training step a target location is presented at a randomly chosen location within the workspace of the robot. A pair of cameras monitors the scene and a preprocessing unit extracts for each camera a pair of coordinates (x, y) that denote the 2D-location of the target in the camera's "retina." The two pairs of camera coordinates are grouped to a four-dimensional vector $\mathbf{u}_{target} = (x_{left}, y_{left}, x_{right}, y_{right})^T$ in the input space.

To be able to position its end effector correctly, the robot system has to know the transformation from $\mathbf{u}_{target}$ to the corresponding set of joint angle motor commands $\vec{\theta}(\mathbf{u}_{target})$. The transformation depends on the geometry of all the robot arm segments as well as on the position and the optical mapping properties of the cameras. The objective is to learn the transformation $\vec{\theta}(\mathbf{u}_{target})$ without an external teacher. In this work we consider the non-redundant case of a robot arm with three degrees of freedom and a rigid wrist (for a more detailed discussion see [7, 11]).

The principle idea behind our approach is the following: the input space is discretized into a set of disjoint cells $\mu \in \{1, 2, \dots N\}$ with centers $\mathbf{w}_\mu$ in the target space, one neural unit being allocated for each cell. Two outputs are assigned to every neural unit, namely a vector $\vec{\theta}_\mu$ and a 3×4-matrix $\mathbf{A}_\mu$ which constitute a linear Taylor expansion of $\vec{\theta}(\mathbf{u}_{target})$ around the discretization point or "reference vector" $\mathbf{w}_\mu$,

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

Both the choice of the discretization cells and the adaptive learning of their output values is done by the "neural-gas" network [9]. This network consists of a set of N neural units, labeled by their index μ and assigned to the location $\mathbf{w}_\mu$. The degree to which each neural unit becomes involved in the current positioning step is determined by the rank of its closeness (in the input space) to the given input stimulus $\mathbf{u}_{target}$. For each new target location we assess the sequence $(\mu_0, \mu_1, \dots, \mu_{N-1})$ of neural units in increasing order of their distance to the input vector $\mathbf{u}_{target}$, defined by

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

The transformation $\vec{\theta}(\mathbf{u}_{target})$ is locally approximated by (1). For the positioning movement, however, not only the linear approximation associated with the unit μ_0 "closest" to $\mathbf{u}_{target}$ determines the joint angles, but the output of a whole subset of neural units with their vectors $\mathbf{w}_\mu$ neighboring to $\mathbf{u}_{target}$ is taken into account. This is achieved by averaging (1) over all neural units μ_k, weighted by a function $g^{mix}(\mu_k)$ which accounts for the "degree of closeness" of $\mathbf{w}_{\mu_k}$ to the input signal $\mathbf{u}_{target}$. The function $g^{mix}(\mu_k) = g^{mix}(k)$ depends on the number k of neural units with their vectors $\mathbf{w}_\mu$ closer to $\mathbf{u}_{target}$ than μ_k. It has its maximum at $g^{mix}(k = 0)$ and decreases to zero as k increases. A convenient choice is $g^{mix}(k) = \exp(-k/\lambda^{mix})$. The joint angles the network produces to reach the target are then given by

$$\vec{\theta}_{initial} = s^{-1} \cdot \sum_k g^{mix}(k) \left(\vec{\theta}_{\mu_k} + \mathbf{A}_{\mu_k}(\mathbf{u}_{target} - \mathbf{w}_{\mu_k})\right), \quad \text{normalized by } s = \sum_k g^{mix}(k). \tag{3}$$

The positioning of the robot arm consists of two phases. The first phase consists of the coarse movement to the initial joint angle set $\vec{\theta}_{initial}$ given by (3). The resulting end effector location seen by the cameras is denoted by $\mathbf{v}_{initial}$. In a second phase the residual deviation from the desired target location $(\mathbf{u}_{target} - \mathbf{v}_{initial})$ is transformed into a correctional movement using the Jacobian $\mathbf{A}(\mathbf{v}_{initial})$. To determine $\mathbf{A}(\mathbf{v}_{initial})$ we take the average of all Jacobian matrices $\mathbf{A}_\mu$, weighted again by g^{mix}. Similar to (3), the neural unit μ_0 with its $\mathbf{w}_{\mu_0}$ closest to $\mathbf{u}_{target}$ contributes the most, for the unit μ_1 with its $\mathbf{w}_{\mu_1}$ second closest to $\mathbf{u}_{target}$ the contribution is second largest, and so forth. This yields the joint angle adjustment

$$\Delta\vec{\theta} = s^{-1} \cdot \sum_k g^{mix}(k) \mathbf{A}_{\mu_k}(\mathbf{u}_{target} - \mathbf{v}_{initial}) \tag{4}$$

and leads the end effector to the final position $\mathbf{v}_{final}$ seen by the cameras.

For a successful operation of the system suitable values for $\mathbf{w}_\mu$, $\vec{\theta}_\mu$ and $\mathbf{A}_\mu$ must be determined. This is done adaptively during the learning phase of the system: after completion of a trial all the neural units are adjusted according to

$$\mathbf{w}_{\mu_k}^{new} = \mathbf{w}_{\mu_k}^{old} + \varepsilon \cdot g(k) \cdot (\mathbf{u}_{target} - \mathbf{w}_{\mu_k}^{old}) \tag{5}$$

$$(\vec{\theta}_{\mu_k}, \mathbf{A}_{\mu_k})^{new} = (\vec{\theta}_{\mu_k}, \mathbf{A}_{\mu_k})^{old} + \varepsilon' \cdot g'(k) \cdot (\Delta\vec{\theta}_{\mu_k}, \Delta\mathbf{A}_{\mu_k}). \tag{6}$$

Here ε and ε' scale the overall size of the adaptation step, and $g(k)$ and $g'(k)$ are, analogous to g^{mix}, functions of the "closeness ranking" index k. They have their maximum at unity for the closest neural unit, i.e. $g_\mu(\mu_0) = g(0) = 1$ and $g'_\mu(\mu_0) = g'(0) = 1$, and decrease to zero as k increases. The effect is an adaptation of a whole subset of neural units in the "neighborhood region" of the closest unit μ_0, thus increasing the rate of convergence. The neighborhood region contains only neural units that have to learn similar parameter values, and, therefore, may profitably participate in the same adjustment step.

A convenient choice is $g(k) = \exp(-k/\lambda)$ and likewise $g'(k) = \exp(-k/\lambda')$. Here λ and λ' determine the size of the neighborhood region. Initially, their value is chosen fairly large for rapid, coarse adaption of many neural units at each adaptation step and gradually decreases for the fine-tuning of the system.

Procedure (5) generates a homogeneous discretization of the relevant submanifold of the input space as explained in more detail in [9]. $\Delta\mathbf{A}_\mu$ in (6) is given by an error correction rule of Widrow-Hoff type [12]

$$\Delta\mathbf{A}_\mu = \left(\Delta\vec{\theta} - \mathbf{A}_\mu\Delta\mathbf{v}\right) \|\Delta\mathbf{v}\|^{-2} \Delta\mathbf{v}^T \tag{7}$$

which minimizes the quadratic cost function

$$E = (\Delta\vec{\theta} - \mathbf{A}_\mu\Delta\mathbf{v})^2 \qquad \text{with} \;\; \Delta\mathbf{v} = \mathbf{v}_{final} - \mathbf{v}_{initial}. \tag{8}$$

To obtain $\Delta\vec{\theta}_\mu$ for the correction of the $\vec{\theta}_\mu$ we employ the locally valid relation $\vec{\theta}_{initial} - \vec{\theta}_\mu^{\,*} = \mathbf{A}_\mu(\mathbf{v}_{initial} - \mathbf{w}_\mu)$ with $\vec{\theta}_\mu^{\,*}$ as the new estimate for the desired value of $\vec{\theta}_\mu$. This yields

$$\Delta\vec{\theta}_\mu = \vec{\theta}_\mu^{\,*} - \vec{\theta}_\mu^{\,old} = \vec{\theta}_{initial} - \vec{\theta}_\mu^{\,old} - \mathbf{A}_\mu(\mathbf{v}_{initial} - \mathbf{w}_\mu) \tag{9}$$

which completes our description of the algorithm.

3. 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 lab to provide a testbed for neural algorithms has to overcome the limitations of today's commercially available robot controllers with respect to computational power and especially to the transparency of their programming languages. In our implementation we chose to employ a Unix workstation to directly control a Westinghouse Robot Puma 562 in real-time via a high speed communication link.

Figure 1 illustrates the main hardware components of our implementation. The Puma 6 DOF manipulator is connected to the Unimation Controller (Mark III, bottom left) which itself contains several controllers. The separate servo controllers for each revolute joint are commanded by a main controller CPU. This CPU usually runs the industrial robot software VAL II which is rather inflexible for our purpose, as it does not support control by an auxiliary computer. To achieve real-time control through an Unix workstation we employed the powerful software package RCCL/RCI (*R*obot-*C*ontrol-*C*-*L*ibrary and *R*eal-time *C*ontrol *I*nterface) [6].

The Unix workstation is a VME based SUN 4/370, which hosts some interfacing hardware and two image processing boards (ICS-400, Androx Inc.), each based on four digital signal processors. These boards provide the computing power for fast data extraction. The sensory input comes from

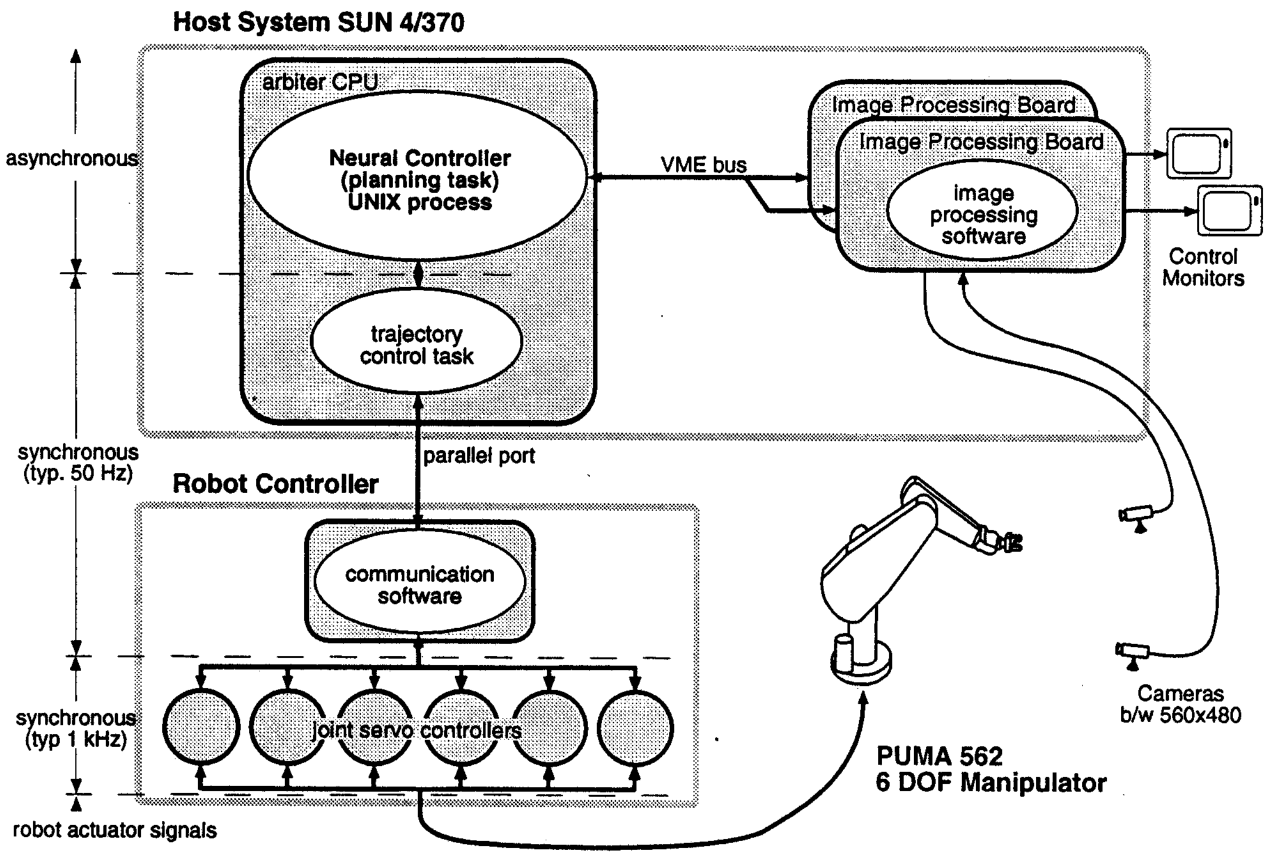


Fig. 1: The main hardware components of the robot system.

two monochrome CCD cameras (560×480 resolution), oriented towards the robot's workspace with a disparity angle of about 50°.

All subsystems are directed by the adaptive control program ("planning task"), written and executed as an ordinary C program in a Unix environment. The program issues motion requests to the trajectory control level. The control task is executed periodically at a high priority and is responsible for reading input data, generating intermediate joint setpoints, and carrying out a "watchdog" function (collision avoidance). During each control cycle (typically 50 Hz) a command package is sent via the parallel port to the robot controller. The main controller CPU is reprogrammed to dispatch commands to the joint servos, collect feedback data and perform elementary safety checks.

To keep the problem of image segmentation simple, during the initial stage of our reseach, we mounted a miniature lamp on the gripper hand (lamp power controlled by software). This object is well defined and the device can be easily discriminated against any background. To identify the location of the lamp in one of the two camera images, the set Π_{max} of the brightest pixels in the video frame is determined, and the center of the minimal rectangle Π_{rect} enclosing Π_{max} is taken to approximate the exact target location in image coordinates. A subsequent consistency test assures identification of the lamp rather than a spurious reflex. It includes checking the geometrical size of Π_{max}, minimal covering ratio of Π_{rect} by Π_{max}, bounded maximal brightness β_{on}^{max}, and, in dubious cases, minimal response $\beta_{on}^{max} - \beta_{off}^{max}$ to turning off the lamp.

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

• *Set target mode:* The target location is given by moving the arm's end effector to a position which is randomly chosen with a uniform distribution in joint angle configuration space. The cameras view the resulting position of the end effector and the arm may return to its previous configuration.

• *Retrieve position from camera information:* After this setting of the target the neural-gas algorithm computes joint angles for the coarse movement, followed by the fine movement to approach the target position more accurately.

Each trial takes on average 8 seconds to complete. The performance of the algorithm is monitored by computing the average Euclidian distance between target and end effector position. This can be done with sufficient accuracy by evaluating the known geometry of the robot and the joint angles, read from the calibrated built-in encoders.

4. Experimental Results

In this section we present 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 values were chosen as $\varepsilon_i = 0.3, \varepsilon_f = 0.05, \varepsilon'_i = \varepsilon'_f = 0.9, \lambda_i = 150, \lambda'_i = \lambda_i^{mix} = 50$ and $\lambda_f = \lambda'_f = \lambda_f^{mix} = 1$. The Jacobians $\mathbf{A}_\mu$ are initialized by assigning a random value from the interval $[\pm\frac{2.7'}{pixel}]$ to each element. A similar procedure was used for the initialization of the $\vec{\theta}_\mu$.

Figure 2 presents a sequence of learning states 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 $\mathbf{w}$ onto the image plane of each camera. Initially, the vectors $\mathbf{w}$ are distributed randomly in the image of the cameras. After about 2000 learning steps the initial distribution has retracted from the four-dimensional input space 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.

A very important advantage of self-learning algorithms is their ability to adapt to different and changing environments. To demonstrate the adaptability of the presented network, we interrupted the learning procedure after 3000 training steps and extended the last arm segment by 100 mm. Figure 3 displays how the algorithm responded.

The thin upper curve shows the positioning capability after the "coarse movement" (open loop system), and the lower bold line indicates the result after the correcting "fine movement" (one time closed loop), both plotted versus the number of adaptation steps. A comparison of the two curves

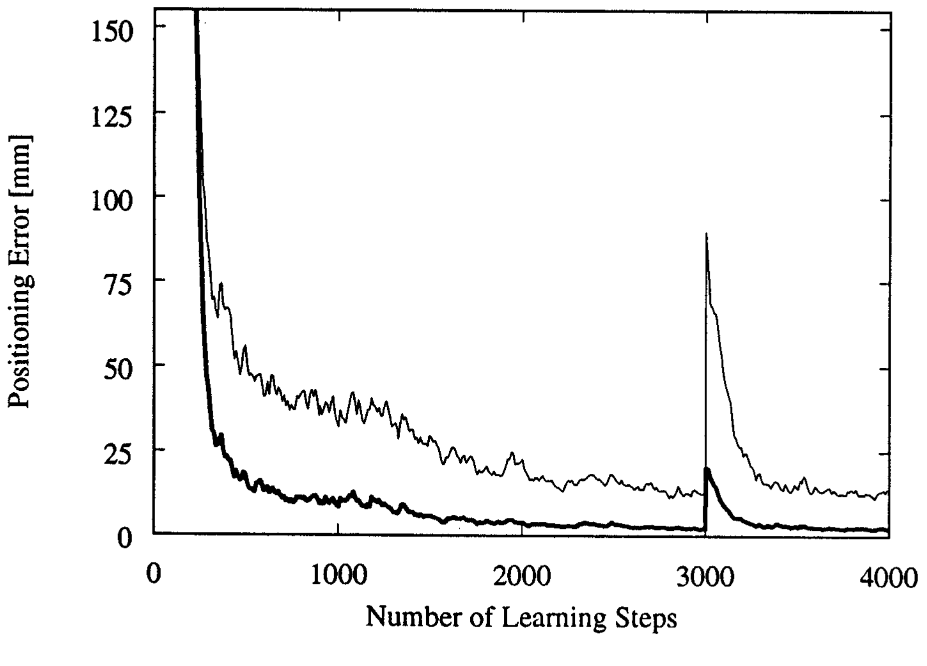
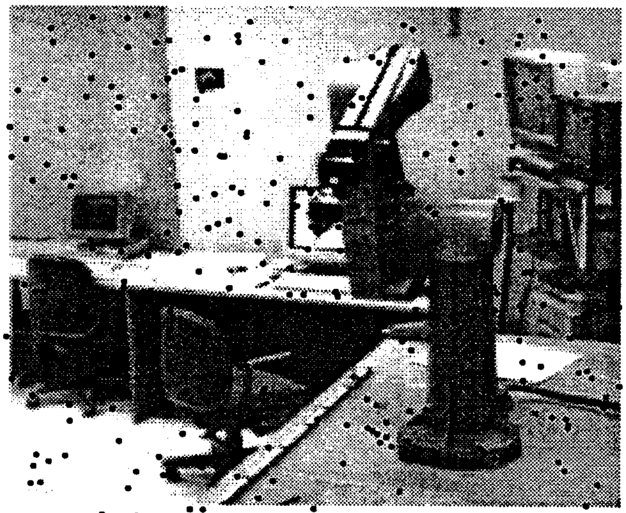
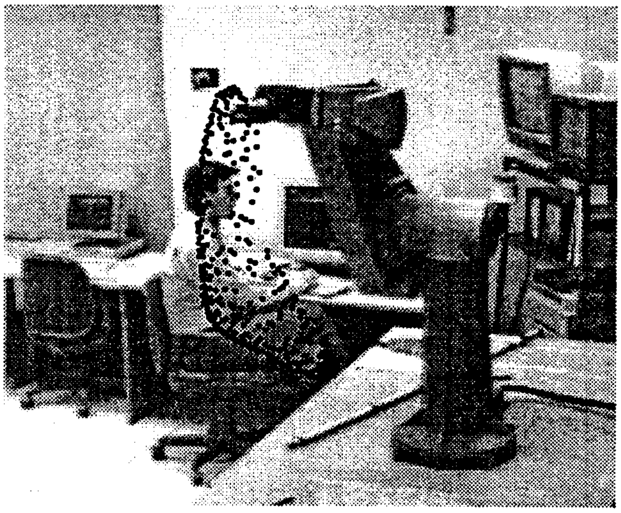
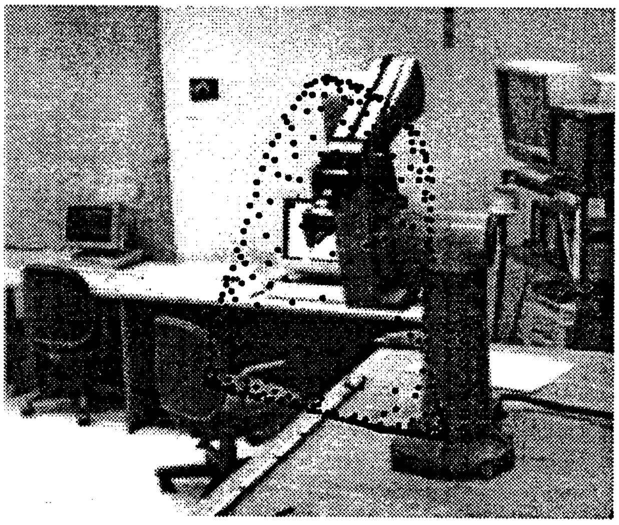
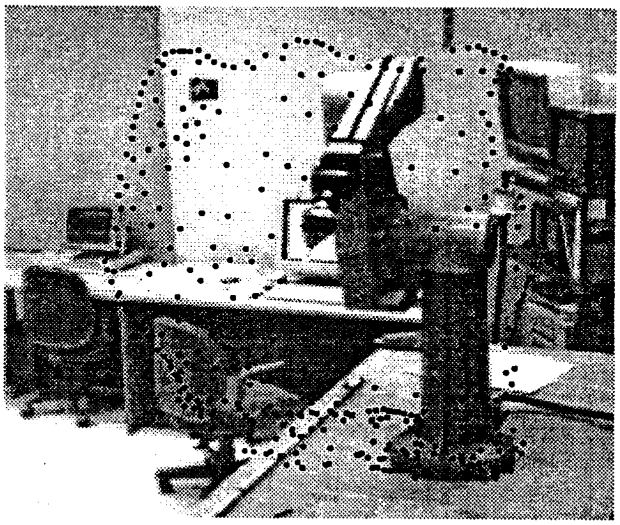


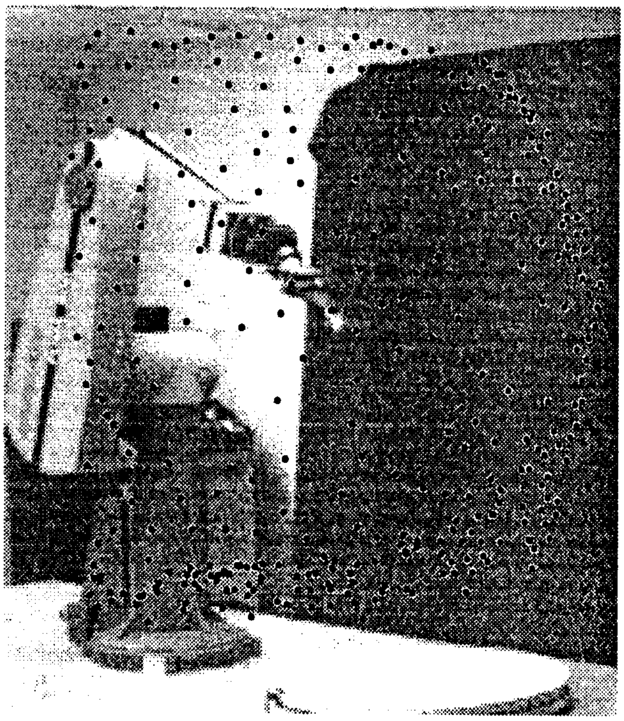
Fig. 3: The positioning error without (coarse movement: thin line), and with visual feedback (fine movement: bold line), plotted over the course of learning. After 3000 learning steps the last arm segment was elongated by 100 mm ($\simeq 10\,\%$).


runs4/net.1.dat -- Camera 1


runs4/net.100.dat -- Camera 1


runs4/net.300.dat -- Camera 1


runs4/net.1500.dat -- Camera 1



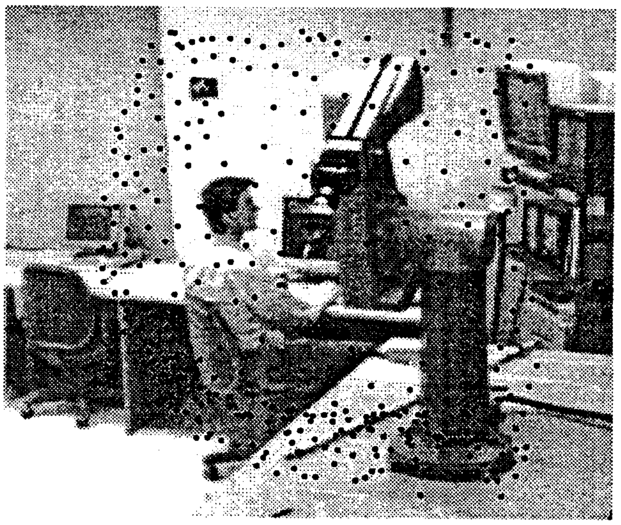


Fig. 2: The development of the network seen as a projection of the reference vectors **w** onto the image planes of the cameras. Shown are the states after t learning steps, with $t = 1$, $t = 100$, $t = 300$, $t = 1500$ (right camera) and $t = t_{max} = 4000$ (both cameras, bottom pictures).

illustrates the substantial improvement of using visual feedback. With the first 3000 trials the error after the fine movement decreased very rapidly to an asymptotic value of 2.2 mm. After the drastic change of the robot's geometry only 300 further iteration steps were necessary to readapt the network for reaching the robot's previous positioning accuracy.

We now want to ask 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 cycles, one regular cycle and one with a non-shared output learning scheme ($\lambda' = 0$).

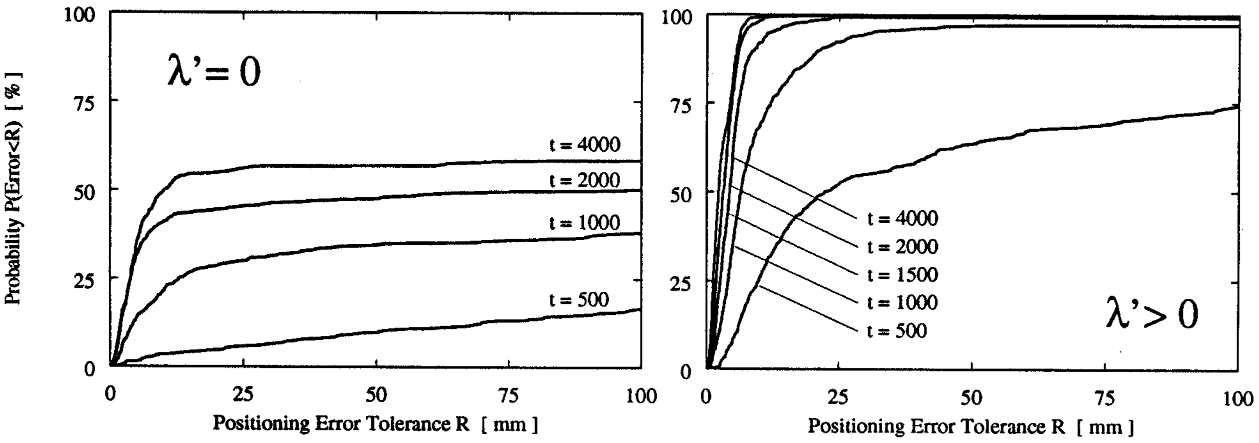


Fig. 4: Positioning ability with (right) and without (left) collective learning of the output values. Plotted is the probability $P_t(R)$ to keep a given error tolerance R at different learning stages t.

When positioning skills are poor — as they are in the the early learning phase, and remain in the latter cycle — 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 can not be recorded by averaging the absolute positioning errors as shown in Figure 3. To obtain a performance measure that also accounts for these faulty trials, we now consider the probability $P_t(R)$ to keep a given error tolerance R at time t. $P_t(R)$ can be written as

$$P_t(R) = \int_0^R p_t(r)dr \quad \text{with} \quad r = \text{Euclidian positioning error}, \tag{10}$$

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

Figure 4 presents at the left side a system of units learning independently ($\lambda' = 0$). The positioning skill, shown at $t = 500$, $t = 1000$, $t = 2000$, and after $t = t_{max} = 4000$, increases rather slowly. The resulting control is still insufficient, each third effort is erroneous. The figure on the right side depicts a system of neural gas units with collective learning among neighboring units. After a few hundred trials the algorithm already performs better than the independent learning system ever achieves. 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 performace.

5. Conclusions

We implemented a new visuomotor control algorithm for the positioning task on an industrial robot system. We were able to achieve a final positioning accuracy of 2.2 mm or 0.2 % of the linear dimension of the workspace of the robot arm. Furthermore, the system succeeded to rapidly adapt to drastically changing situations. The accuracy is currently limited by the image processing resolution and not by the control algorithm. Yet, this algorithm achieves a precision that is higher by one order of magnitude than earlier neural network implementations, like Kuperstein's system with 4 % average deviation [5]. Compared to the predecessor version [7] of the introduced learning algorithm a roughly ten-fold learning rate was accomplished. This reduction of the required number of trial movements is partly due to the employment of the "neural-gas" network, which saves the learning steps which are necessary for the initial topological ordering of the neural units of the Kohonen network used in [7]. Learning steps are shared by a whole subset of neural units, determined by their neighborhood relation in the input space rather than indirectly by their relative location in a prestructured array. We found that the collective learning greatly enhances the algorithm's performance with respect to robustness as well as the rate of convergence.

6. Acknowledgement

We would like to thank Narendra Ahuja for providing the Puma robot for this project. This work has been supported by the Beckman Institute for Advanced Science and Technology of the University of Illinois at Urbana-Champaign, by the National Science Foundation under Grant DIR-90-15561, and by a fellowship of the Volkswagen Foundation to T.M.

References

[1] R. Eckmiller, J. Beckmann, H. Werntges, and M. Lades. Neural kinematics net for a redundant robot arm. In *IJCNN, Washington DC*, volume 2, pages 333—340, 1989.

[2] T. Kohonen. Self-organized formation of topographically correct feature maps. *Biol. Cybern.*, 43:59–69, 1982.

[3] T. Kohonen. *Self-organization and Associative Memory*, volume 8 of *Springer Series in Information Science*. Springer, Berlin, Heidelberg, New York, 1984.

[4] M. Kuperstein. Adaptive visual-motor coordination in multijoint robots using parallel achitecture. In *IEEE Int. Automat. Robotics (Raleigh, NC)*, pages 1596–1602, 1987.

[5] M. Kuperstein. Neural model of adaptive hand-eye coordination for single postures. *Science*, 239:1301–1311, 1988.

[6] 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)*, pages 465–469, April 1988.

[7] T. Martinetz, H. Ritter, and K. Schulten. Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Trans. Neural Networks*, 1(1):131–136, March 1990.

[8] T. Martinetz and K. Schulten. Hierarchical neural net for learning control of a robot's arm and gripper. In *Proc. IJCNN-90, San Diego*, volume 3, pages 747–752, June 1990.

[9] T. Martinetz and K. Schulten. A "neural gas" network learns topologies. In *Proc. ICANN, Helsinki*, 1991.

[10] B.W. Mel. A robot that learns by doing. In *AIP Neural Information Processing System Conf, Denver, CO*, 1987.

[11] J. Walter, T. Martinetz, and K. Schulten. Visuomotor control on an industrial robot, using "neural gas" algorithm. *(in preparation)*, 1991.

[12] B. Widrow and M.E. Hoff. Adaptive switching circuits. *WESCON Conv Record*, 4:96–104, 1960.