11. VISUOMOTOR COORDINATION OF A ROBOT ARM

In this chapter we again use the extension of Kohonen's model that includes the learning of output values. By employing this extended model we will enable a robot system to learn to automatically position its end effector at a target that has been presented in the robot's work space (Ritter, Martinetz, and Schulten 1989; Martinetz, Ritter, and Schulten 1989, 1990a, 1990b). "End effector" is the name given in robotics to a tool at the end of the robot arm, *e.g.*, a gripper, a welding electrode, etc. End effector positioning is an integral part of almost any task with which a robot system might be confronted. Examples of tasks include grasping of objects, application of welding points, insertion of devices, or, to mention a task from robotics that has not yet been satisfactorily accomplished, the planning of trajectories to circumvent obstacles. Obviously, end effector positioning is fundamental for all robot tasks and, therefore, we turn to this problem first.

Figure 11.1 shows a robot system as it has been simulated in the computer. The robot consists of a triple-jointed arm, positioned in front of the work space. The arm can pivot around its base (θ_1), and the other two joints (θ_2, θ_3) allow the arm to move in a vertical plane. For successful operation the robot requires information about the location of the targets. Humans obtain this information through their eyes. Correspondingly, we equip our robot with two cameras which can observe the work space. It is important to use *two* cameras in order to perceive the three-dimensionality of the space.

Since the work space of the robot is three-dimensional, we will now use, in contrast to examples in previous chapters, a three-dimensional Kohonen net, *i.e.*, a Kohonen lattice. With this step we seem to deviate from the networks actually realized in the brain, which appear to have a two-dimensional topology, because, as we know, the cortex consists of a two-dimensional arrangement of neural functional units, the so-called micro-columns. (see, *e.g.*, Kandel and Schwartz 1985). But this discrepancy is only an apparent one; the actual relevant topology, given by the connecting structure of the

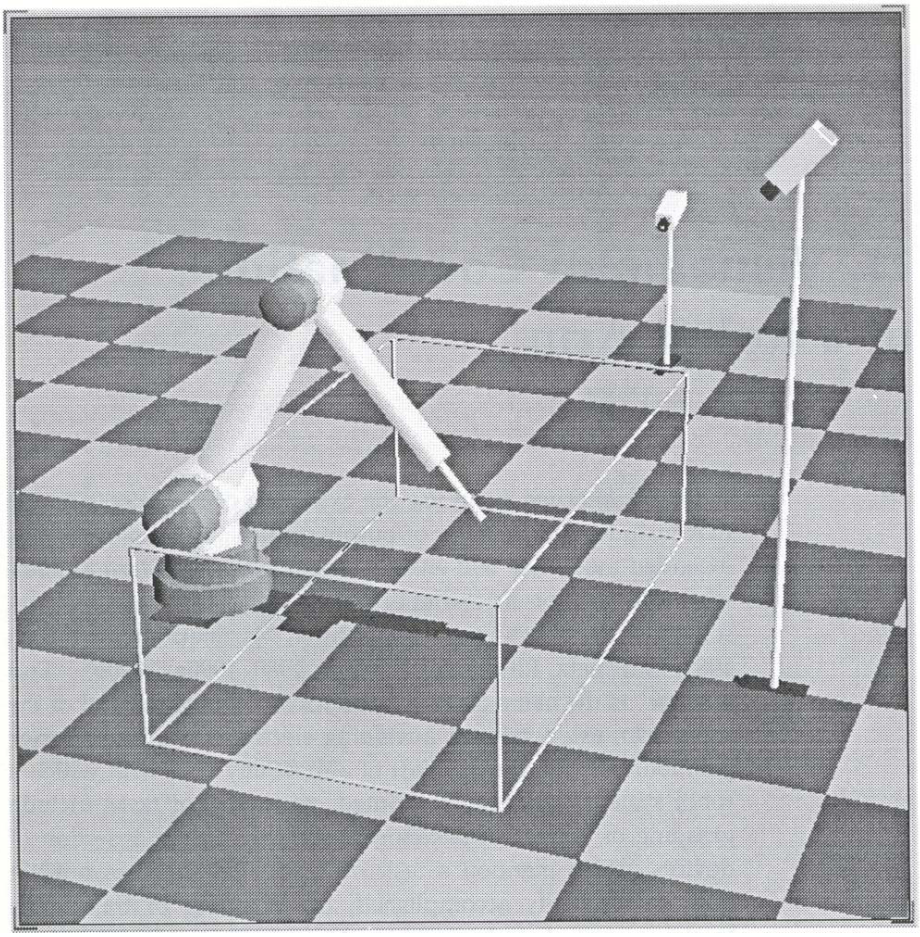

Abb. 11.1: Model of the simulated robot system. The arm has three degrees of freedom: rotation around the vertical axis (θ_1), a middle joint (θ_2), and an outer joint (θ_3). The axes of the middle joint and the outer joint are parallel to each other, and are both horizontal and perpendicular to all three arm segments. Camera 1 is in front of the work space, and camera 2 is located on the left side of the robot.

neurons or functional units, can very well deviate from the morphological arrangement on the cortex. Suppose we take a three-dimensional wire lattice and press it into a two-dimensional layer. The connecting structure between the lattice nodes has of course not been changed by this modification; it

is still three-dimensional in spite of the fact that all lattice nodes now lie in a plane. Accordingly, the connecting structure between neural units in the brain might be multi-dimensional even if the neurons are arranged in a two-dimensional layer. The three-dimensional topology does not change any essential features of Kohonen's algorithm. Only the lattice vectors $\mathbf{r}$ are three-dimensional, and the distance $\|\mathbf{r} - \mathbf{s}\|$ to the excitation center is now measured in a three-dimensional lattice.
During the training phase, the position of the targets in the work space are chosen randomly. Before each movement the target is viewed by the cameras, the signals of which are fed into the neural net. Each neural unit is responsible for a particular region observed by the cameras. For each incoming signal that neuron which is momentarily responsible for the location of the target becomes activated and transmits its output values to the motor control. The three-dimensional coordinates of the location of the target, however, are not directly available; each camera only delivers the location at which the target appears in its two-dimensional visual field. The network must then derive proper output values for the joint motors to position the end effectors properly.
The neural network does not receive any prior information about, for example, the location of the cameras or the lengths of the robot arm segments. Rather it has to learn these geometrical relationships in order to correctly convert the camera information into motor signals. For this reason the untrained arm will initially not reach most targets. In each trial the deviation is observed by the cameras and then used to improve the output values. At each successive step a new target point is presented to the robot, providing the opportunity for further learning. The robot represents an autonomous system that works in a closed-loop mode and performs completely without a teacher. The robot system receives all the information needed for adaptation from its own stereo cameras and, thus, learns without an external teacher (See also Ginsburg and Opper 1969; Barto and Sutton 1981; Kuperstein 1987, 1988; Miller 1989).

11.1 The Positioning Action

As displayed schematically in Fig. 11.2, each target point generates an image point on the image plane of each camera. The two-dimensional position vectors $\vec{u}_1$, $\vec{u}_2$ of the image points in the image planes of cameras 1 and 2

implicitly transmit to the system the spatial position of the target point which is uniquely defined by $(\vec{u}_1, \vec{u}_2)$. We combine both vectors to one four-dimensional input signal $\mathbf{u} = (\vec{u}_1, \vec{u}_2)$. In order to be able to correctly position its end effector, the robot system must be able to perform the transformation $\vec{\theta}(\mathbf{u})$ from image point coordinates $\mathbf{u}$ of the target to the required set of joint angles $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$ of the arm. This transformation depends on the geometry of the robot arm as well as on the location and imaging characteristics of the cameras, and should be adapted automatically through the learning method.

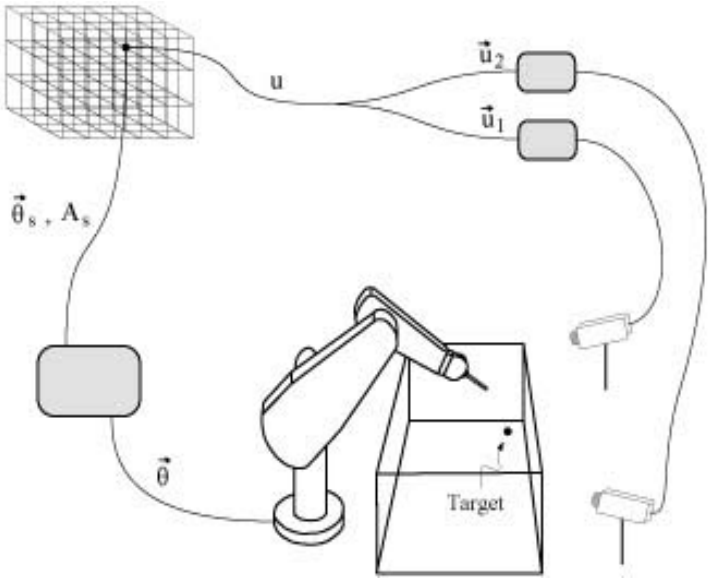

Abb. 11.2: Schematic diagram of the positioning action. The two-dimensional coordinates $\vec{u}_1$ and $\vec{u}_2$ of the target in the image planes of cameras 1 and 2 are combined to a four-dimensional vector $\mathbf{u} = (\vec{u}_1, \vec{u}_2)$ and then transmitted to the three-dimensional Kohonen net. The neural unit $\mathbf{s}$ which is responsible for the region in which the target is momentarily located is activated and makes available its two output elements, the expansion terms of 0-th and first order, $\vec{\theta}_\mathbf{s}$ and $\mathbf{A_s}$. These terms determine the joint angles needed for the movement towards the target.

For each target $\mathbf{u}$ the neural unit whose receptive field entails the target location responds. As before, the receptive fields are defined by synaptic

strengths $\mathbf{w_r}$ that now consist of four components as does $\mathbf{u}$. A neural unit $\mathbf{s}$ is responsible for all targets $\mathbf{u}$ for which the condition $\|\mathbf{w_s} - \mathbf{u}\| \leq \|\mathbf{w_r} - \mathbf{u}\|$ holds, where $\mathbf{r}$ denotes all neurons on the lattice that are different from $\mathbf{s}$. The responding neural unit provides as output values a suitable set of joint angles $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$ that is transmitted to the three joint motors and that should lead the end effector to the target. To generate $\vec{\theta}$, two output elements that must be learned are assigned to each neuron: a three-dimensional vector $\vec{\theta}_\mathbf{r}$ and a 3×4-matrix $\mathbf{A_r}$. At the end of the learning phase, when $\vec{\theta}_\mathbf{r}$ and $\mathbf{A_r}$ have taken on the desired values, the angle positions $\vec{\theta}_\mathbf{r}$ will lead the end effector to the center of the receptive field of neuron $\mathbf{r}$; *i.e.*, $\vec{\theta}_\mathbf{r}$ defines the move of the end effector into the target position $\mathbf{u} = \mathbf{w_r}$. The Jacobian matrix $\mathbf{A_r} = \delta\vec{\theta}/\delta\mathbf{u}$ serves to linearly correct the joint angles if the input vector $\mathbf{u}$ does not coincide with $\mathbf{w_r}$. This correction is accomplished by a linear expansion around $\mathbf{w_r}$ which is restricted to the region of responsibility of the particular neural unit $\mathbf{r}$. The angular configuration $\vec{\theta}$ that is transmitted to the joints by unit $\mathbf{s}$ is then given by

$$\vec{\theta} = \vec{\theta}_\mathbf{s} + \mathbf{A_s}(\mathbf{u} - \mathbf{w_s}). \tag{11.1}$$

At the end of the learning process this expression corresponds to a linear expansion around $\mathbf{w_s}$ of the exact transformation $\vec{\theta}(\mathbf{u})$. The vector $\vec{\theta}_\mathbf{s}$ is given by the expansion term of zeroth order, and $\mathbf{A_s}$ is given by the Jacobian matrix for $\vec{\theta}(\mathbf{u})$ at $\mathbf{w_s}$.

The values $\vec{\theta}_\mathbf{r}$ and $\mathbf{A_r}$ should assume values $\vec{\theta}_\mathbf{r}{}^0$, $\mathbf{A}_\mathbf{r}^0$ in the course of the learning which minimize the average output error. By the use of Eq. (11.1) the exact transformation of $\vec{\theta}(\mathbf{u})$ is approximated by an adaptive covering of the domain of $\vec{\theta}(\mathbf{u})$ with locally valid linear mappings. When compared with the sole use of discrete output values, the introduction of $\mathbf{A_r}$ implies that maps with less discretization points (neural units) are sufficient to reach a given precision.

The two vectors $(\vec{u}_1, \vec{u}_2)$, combined into the single vector $\mathbf{u}$, span a four-dimensional space. All the target positions lie within the three-dimensional work space as indicated in Figs. 11.1 and 11.2. Hence, the input signals $\mathbf{u}$ of the cameras are all located in a three-dimensional submanifold of the four-dimensional input signal space, making it unnecessary to use a four-dimensional Kohonen net to represent the input signal space—a Kohonen net with a three-dimensional lattice topology is sufficient. If we knew the relevant submanifold, *i.e.*, if we had precise knowledge about the position

of the image planes relative to the work space, then we could, with the initialization, "plant" the nodes of the neural net into the submanifold, and the learning algorithm, as before, would in principle only have to "unfold" the net onto this submanifold.

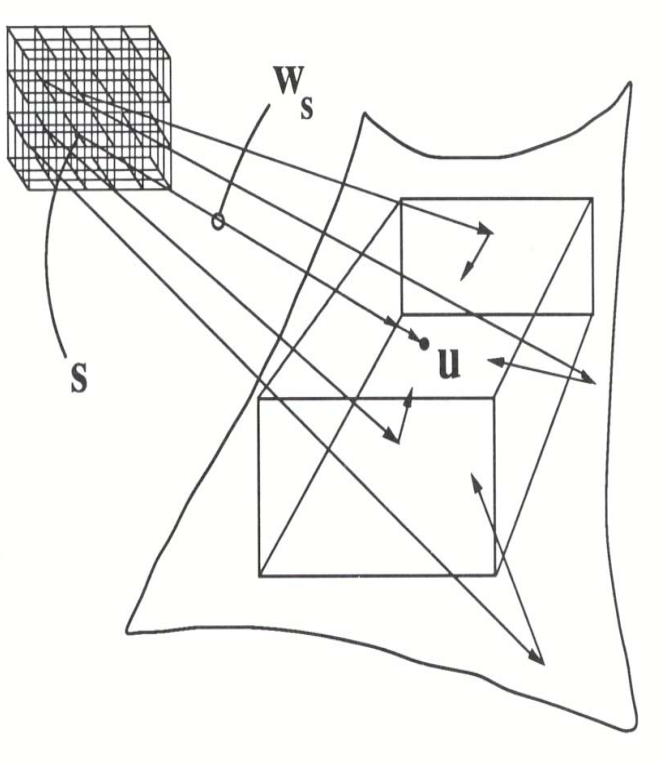

Abb. 11.3: Occupation of the three-dimensional subspace with discretization points. The four-dimensional input signal space, U, schematically displayed, is initially homogeneously filled with the elements of the three-dimensional Kohonen net. The input signals $\mathbf{u}$ lie exclusively within the three-dimensional submanifold W. Therefore, all elements move rapidly into the relevant subspace of U, and a "waste" of unused discretization points W is avoided.

However, supplying any a priori knowledge about the system to the learning algorithm, *e.g.*, precise information about the location of the cameras, should be avoided. Our goal is for the robot system to find out such information by itself while it is learning. Only then are the desired adaptive capabilities of

an algorithm available to enable the robot to adapt to possible changes in the precise system data, *e.g.*, caused by corrosion, wear of its parts, or camera shifts. In our model this means that the robot system must find for itself the relevant three-dimensional submanifold in the space $U = U_1 \otimes U_2$ of the two connected image planes U_1, U_2 in order to distribute as homogeneously as possible the lattice points that are necessary for the discretization.
Lacking a priori knowledge about the location of the relevant subspace, we distribute the nodes of the neural lattice randomly within the four-dimensional input signal space at the beginning of the learning phase. The positions of the lattice nodes in the space U are described as before by four-dimensional vectors $\mathbf{w_r}$. Because the incoming input stimuli $\mathbf{u} = (\vec{u}_1, \vec{u}_2)$ all lie within the yet unknown submanifold, just a few learning steps cause a "contraction" of all net nodes onto this relevant subspace. This contraction is schematically displayed in Fig. 11.3
The representation of only the submanifold by the network is a direct consequence of the feature of Kohonen's algorithm that has been discused extensively in Chapter 5, *i.e.*, adaptation of the density of the net nodes according to the probability density of the input signals. Since the input probability distribution in the present case differs from zero only on the submanifold, it follows that in equilibrium the density of the net nodes will be zero outside the submanifold. In this way the robot system can discover by itself the subspace relevant for its functioning. In the ensuing course of learning, the three-dimensional net will unfold within this three-dimensional subspace and will homogeneously distribute its nodes as discretization points.
This again demonstrates the effectiveness of the model in using a finite number of neural units, or more generally, memory units. We could have initialized the total, four-dimensional space with rigid discretization points. With this naive approach, the number of memory elements required to achieve a suitable representation would be much higher because the number of required memory elements increases exponentially with the dimensionality of the space. If we used a 20×20×20 lattice in our model, then the memory requirement to fill the total four-dimensional space with the same density of discretization points would be greater by at least a factor of twenty. In most cases the additional memory requirement would be even higher since the actual work space spans only part of the visual field of each camera.
The output values are learned by analysing the positioning error. The positioning of the arm occurs in two steps. If $\mathbf{s}$ is the node which responds to the target $\mathbf{u}$, then in a first step the positioning of the end effector is achieved by

means of the linear approximation (11.1) of the exact transformation $\vec{\theta}(\mathbf{u})$. The resulting position of the angle is denoted by $\vec{\theta}_i$. This first step causes the joint angles to assume a position given by

$$\vec{\theta}_i = \vec{\theta}_\mathbf{s} + \mathbf{A}_\mathbf{s}(\mathbf{u} - \mathbf{w}_\mathbf{s}). \tag{11.2}$$

The image point coordinates $\mathbf{v}_i$ of the end effector corresponding to $\vec{\theta}_i$ are recognized by the cameras and are then used for a second step, a corrective movement.

This correction is achieved in the following way: if the end effector is in the vicinity of the target after the first step of the positioning process, then the expression $\|\mathbf{u}-\mathbf{v}_i\|$ is already sufficiently small and, to a good approximation, $\vec{\theta}_{goal} - \vec{\theta}_i = \mathbf{A}_\mathbf{s}^0(\mathbf{u}-\mathbf{v}_i)$, where $\vec{\theta}_{goal}$ is the angle configuration needed to reach the target. Thus the corrective movement is simply given by the change in joint angles

$$\Delta\vec{\theta} = \mathbf{A}_\mathbf{s}(\mathbf{u} - \mathbf{v}_i). \tag{11.3}$$

The resulting image point coordinates $\mathbf{v}_f$ of the final position of the end effector are again recognized by the cameras and, along with $\mathbf{v}_i$, are put into a subsequent adaptation step. The corrective movement can be iterated several times, thereby reducing the positioning error as much as desired as far as imperfections of the equipment permit this. Satisfying results can often be obtained with just a single or very few corrective steps. In the following description we assume, therefore, two positioning steps, a gross movement described by (12.2) and a corrective movement described by (12.3).

11.2 The Learning Method

The learning algorithm for obtaining suitable elements $\vec{\theta}_\mathbf{r}$ and $\mathbf{A}_\mathbf{r}$ for each unit $\mathbf{r}$ uses a gradient descent on a quadratic error function. In each learning step the gradients of the error functions are calculated from the positioning error in order to obtain the direction to the minimum. If neural unit $\mathbf{s}$ was responsible for generating the arm movements, the improved values of $\vec{\theta}_\mathbf{s}$ and $\mathbf{A}_\mathbf{s}$ are then given

$$\begin{aligned} \vec{\theta}^{\,*} &= \vec{\theta}_\mathbf{s}^{\,old} + \delta_1 \cdot \mathbf{A}_\mathbf{s}^{\,old}(\mathbf{u} - \mathbf{v}_i) \\ \mathbf{A}^* &= \mathbf{A}_\mathbf{s}^{old} + \delta_2 \cdot \mathbf{A}_\mathbf{s}^{old}(\mathbf{u} - \mathbf{v}_f)\Delta\mathbf{v}^T, \end{aligned} \tag{11.4}$$

where $\Delta\mathbf{v} = \mathbf{v}_f - \mathbf{v}_i$, $\mathbf{v}_f$ and $\mathbf{v}_i$ as defined in Section 11.1. A derivation of these learning rules will be given in Section 11.3. The factors δ_1 and δ_2

denote the step size of the gradient descent. On the one hand the step sizes should not be too small, for otherwise the number of iterations would be unnecessarily high. On the other hand, they should not be too large, for then a learning step could overshoot the minimum. As we will see later in a mathematical analysis of the learning method, the values $\delta_1 = 1$ and $\delta_2 = 1/\|\Delta\mathbf{v}\|^2$ are optimal.
The new estimates $\vec{\theta}^{\,*}$ and $\mathbf{A}^*$ that were obtained by the gradient descent (11.4) are used to improve the output elements of neural unit $\mathbf{s}$ as well as those of its neighbors. Here, as in previous chapters, we employ a learning procedure for the output elements that is analogous to Kohonen's original algorithm and is given by

$$\begin{aligned} \vec{\theta}_{\mathbf{r}}^{\,\text{new}} &= \vec{\theta}_{\mathbf{r}}^{\,\text{old}} + \epsilon' h'_{\mathbf{rs}}(\vec{\theta}^{\,*} - \vec{\theta}_{\mathbf{r}}^{\,\text{old}}) \\ \mathbf{A}_{\mathbf{r}}^{\text{new}} &= \mathbf{A}_{\mathbf{r}}^{\text{old}} + \epsilon' h'_{\mathbf{rs}}(\mathbf{A}^* - \mathbf{A}_{\mathbf{r}}^{\text{old}}). \end{aligned} \tag{11.5}$$

This learning step modifies a whole population of neural units in the vicinity of unit $\mathbf{s}$. As a following display of simulation results will show, the cooperation between neighboring neural units resulting from Eq. (11.5) is crucial for rapid learning and for convergence to a satisfactory final state. Without cooperation between neighbors, some output elements $\vec{\theta}_{\mathbf{r}}$ and $\mathbf{A}_{\mathbf{r}}$ might not converge towards their desired values, and could become "stuck" in local minima during the gradient descent in a way similar to what occurred in the example of oculomotor control. A more detailed mathematical analysis of this behavior will be given in Chapter 15.
Our learning algorithm for the Kohonen net and the output elements $\vec{\theta}_{\mathbf{r}}$ and $\mathbf{A}_{\mathbf{r}}$ can be summarized into eight steps as follows:

1. Present a randomly chosen target point in the work space.
2. Let the cameras observe the corresponding input signal $\mathbf{u}$.
3. Determine the lattice point (neural unit) $\mathbf{s} := \phi_{\mathbf{w}}(\mathbf{u})$ to which $\mathbf{u}$ is assigned in the lattice.
4. Move the end effector to an intermediate position by setting the joint angles to

$$\vec{\theta}_i = \vec{\theta}_{\mathbf{s}} + \mathbf{A}_{\mathbf{s}}(\mathbf{u} - \mathbf{w}_{\mathbf{s}}),$$

and register the corresponding coordinates $\mathbf{v}_i$ of the end effector in the image planes of the cameras.

5. Execute a correction of the end effector position from step 5 according to

$$\vec{\theta}_f \;=\; \vec{\theta}_i + \mathbf{A_s}(\mathbf{u} - \mathbf{v}_i),$$

and observe the corresponding camera coordinates $\mathbf{v}_f$.

6. Execute the learning step for the receptive field of $\mathbf{r}$ according to

$$\mathbf{w}_\mathbf{r}^{\text{new}} \;=\; \mathbf{w}_\mathbf{r}^{\text{old}} + \epsilon h_{\mathbf{rs}}(\mathbf{u} - \mathbf{w}_\mathbf{r}^{\text{old}}).$$

7. Determine improved values $\vec{\theta}^{\,*}$ and $\mathbf{A}^*$ using

$$\begin{aligned}\vec{\theta}^{\,*} &= \vec{\theta}_\mathbf{s}^{\,old} + \delta_1 \cdot \mathbf{A_s}^{old}(\mathbf{u} - \mathbf{v}_i)\\ \mathbf{A}^* &= \mathbf{A}_\mathbf{s}^{old} + \delta_2 \cdot \mathbf{A}_\mathbf{s}^{old}(\mathbf{u} - \mathbf{v}_f)(\mathbf{v}_f - \mathbf{v}_i)^T.\end{aligned}$$

8. Execute a learning step for the output values of the neural unit $\mathbf{s}$ as well as of its neighbors $\mathbf{r}$

$$\begin{aligned}\vec{\theta}_\mathbf{r}^{\text{new}} &= \vec{\theta}_\mathbf{r}^{\text{old}} + \epsilon' h'_{\mathbf{rs}}(\vec{\theta}^{\,*} - \vec{\theta}_\mathbf{r}^{\text{old}})\\ \mathbf{A}_\mathbf{r}^{\text{new}} &= \mathbf{A}_\mathbf{r}^{\text{old}} + \epsilon' h'_{\mathbf{rs}}(\mathbf{A}^* - \mathbf{A}_\mathbf{r}^{\text{old}})\end{aligned}$$

and continue on with step 1.

The second phase of the positioning process (steps 5–8), aside from the correction of the end effector position that resulted from the first motion phase, generates pairs of camera coordinates $\mathbf{v}_i$ and $\mathbf{v}_f$ for the learning of $\mathbf{A_r}$ and $\vec{\theta}_\mathbf{r}$. The Jacobian matrices $\mathbf{A_r}$ describe the relation between a small change of the joint angles and the corresponding change in position of the end effector in the camera coordinates. The generally small corrective movements therefore deliver value pairs that must be connected by the Jacobian matrices that are to be learned, and thus, the corrective movements are used in Eq. (11.4) to iteratively improve the Jacobian matrices $\mathbf{A_r}$.
As soon as the Jacobian matrices have been learned, they can be used to improve the expansion terms of zeroth order $\vec{\theta}_\mathbf{r}$. Each term should lead the end effector in camera coordinates towards the corresponding discretization point $\mathbf{w_r}$. The corresponding error, as seen through the cameras, can be evaluated by the Jacobian matrices to obtain a suitable correction of $\vec{\theta}_\mathbf{r}$ which then results in an improved value $\vec{\theta}^{\,*}$ in learning step (11.4).
Thus, the Jacobian matrices $\mathbf{A_r}$ contribute significantly to three important aspects of the learning algorithm:

1. Calculation of the joint angle changes for the corrective step in order to
   (a) reduce the positioning error, and
   (b) generate a pair $(\mathbf{v}_i, \mathbf{v}_f)$ for an adaptation step of $\mathbf{A_r}$ itself.
2. Improvement of the expansion terms of zeroth order by evaluating the error seen through the cameras to obtain a correction for $\vec{\theta}_\mathbf{r}$.

The Jacobian matrix $\mathbf{A_r}$ is a most essential element of the presented algorithm for learning and controlling the kinematics of the robot arm. In the next section we present a mathematical derivation of the learning algorithm to further illuminate the crucial role of the Jacobian $\mathbf{A_r}$ during the learning process.

11.3 A Derivation of the Learning Method

We now want to substantiate mathematically the method for determining the estimates $\vec{\theta}^{\,*}$ and $\mathbf{A}^*$. For this purpose we consider the neural lattice to be unfolded and stabilized to the extend that the space of input vectors is well represented by the network nodes and that larger shifts of the discretization points $\mathbf{w_r}$ no longer occur. We can then assume the location of the discretization points to be sufficiently constant and can neglect their change by the Kohonen algorithm, so that in the following we only need to consider the behavior of the output elements $\vec{\theta}_\mathbf{r}$ and $\mathbf{A_r}$.
If the end effector positions $\mathbf{v}_f$, $\mathbf{v}_i$, as provided by the cameras, lie sufficiently close to the currently implicated discretization point $\mathbf{w_s}$, then the linear relation

$$\vec{\theta}_f - \vec{\theta}_i = \mathbf{A}_\mathbf{s}^0(\mathbf{v}_f - \mathbf{v}_i). \tag{11.6}$$

is approximately satisfied. The matrix $\mathbf{A}_\mathbf{s}^0$ in this equation is to be obtained by the learning algorithm using the pairs $(\vec{\theta}_i, \vec{\theta}_f)$ and $(\mathbf{v}_i, \mathbf{v}_f)$.
Knowing $\mathbf{A}_\mathbf{s}^0$, we can calculate the 0-th order expansion term $\vec{\theta}_\mathbf{s}^{\,0} = \vec{\theta}(\mathbf{w_s})$ since the linear relation

$$\vec{\theta}(\mathbf{w_s}) - \vec{\theta}(\mathbf{v}_i) = \mathbf{A}_\mathbf{s}^0(\mathbf{w_s} - \mathbf{v}_i) \tag{11.7}$$

holds in a sufficiently small vicinity around $\mathbf{w_s}$. Since $\vec{\theta}(\mathbf{v}_i) = \vec{\theta}_i$ is given by Eq. (11.2) from the positioning process of the joint angles, we obtain for $\vec{\theta}_\mathbf{s}^{\,0}$

the expression

$$\vec{\theta}_{\mathbf{s}}^{\,0} = \vec{\theta}_{\mathbf{s}} + \mathbf{A}_{\mathbf{s}}(\mathbf{u} - \mathbf{w}_{\mathbf{s}}) + \mathbf{A}_{\mathbf{s}}^{0}(\mathbf{w}_{\mathbf{s}} - \mathbf{v}_i). \tag{11.8}$$

Taking

$$\vec{\theta}^{\,*} = \vec{\theta}_{\mathbf{s}} + \mathbf{A}_{\mathbf{s}}(\mathbf{u} - \mathbf{v}_i) \tag{11.9}$$

as an improved estimate for $\vec{\theta}_{\mathbf{s}}$ (see Eq.(11.4)) leads to $\vec{\theta}^{\,*} \rightarrow \vec{\theta}_{\mathbf{s}}^{\,0}$ with $\mathbf{A}_{\mathbf{s}} \rightarrow \mathbf{A}_{\mathbf{s}}^{0}$. Therefore, it is sufficient to develop an algorithm which provides the correct Jacobian matrix $\mathbf{A}_{\mathbf{s}}^{0}$ for every network node $\mathbf{s}$, since then learning step (11.9) is able to provide us with the correct zeroth order expansion terms $\vec{\theta}_{\mathbf{s}}^{\,0}$.

In the learning phase the points $(\vec{\theta}_i, \mathbf{v}_i)$, $(\vec{\theta}_f, \mathbf{v}_f)$, and every target point $\mathbf{u}$ are elements of a whole sequence $(\vec{\theta}_i^{\nu}, \mathbf{v}_i^{\nu})$, $(\vec{\theta}_f^{\nu}, \mathbf{v}_f^{\nu})$ and $\mathbf{u}^{\nu}$ of training data where $\nu = 1, 2, 3, \ldots$.[1] In the following we will consider a single lattice point $\mathbf{s}$, and ν will index only that part of the sequence of training data that leads to an improvement of the output values of $\mathbf{s}$.

In principle it is possible to calculate $\mathbf{A}_{\mathbf{s}}^{0}$ from $\mathbf{v}_i$, $\mathbf{v}_f$ and $\vec{\theta}_i$, $\vec{\theta}_f$ in Eq. (11.6) by using the method of least mean square error. The advantage of this method is that, in terms of the only approximate linear relation (11.6), the mean square error given by

$$E(\mathbf{A}_{\mathbf{s}}) = \frac{1}{2} \sum_{\nu} \left[(\vec{\theta}_f^{\nu} - \vec{\theta}_i^{\nu}) - \mathbf{A}_{\mathbf{s}}(\mathbf{v}_f^{\nu} - \mathbf{v}_i^{\nu}) \right]^2 \tag{11.10}$$

can be minimized for all training data simultanously. Concerning the adaptivity of the system, however, such a procedure has the disadvantage that first the quantities $(\vec{\theta}_i^{\nu}, \mathbf{v}_i^{\nu})$, $(\vec{\theta}_f^{\nu}, \mathbf{v}_f^{\nu})$ need to be accumulated before a result becomes available. After that $\mathbf{A}_{\mathbf{s}}$ is fixed, and the system can adapt to later slow variations of the relation $\vec{\theta}(\mathbf{u})$ only by a complete re-evaluation of $\mathbf{A}_{\mathbf{s}}$. Moreover, a criterion would be needed to decide if a re-evaluation of $\mathbf{A}_{\mathbf{s}}$ is necessary. To avoid these disadvantages we opt for an iterative method that improves an existing approximation $\mathbf{A}_{\mathbf{s}}(t)$ of $\mathbf{A}_{\mathbf{s}}^{0}$ for every new value pair $(\vec{\theta}_i^{\nu}, \mathbf{v}_i^{\nu})$, $(\vec{\theta}_f^{\nu}, \mathbf{v}_f^{\nu})$. A suitable iterative algorithm employes the technique of linear regression and was suggested for application in adaptive systems by Widrow and Hoff (1960).

[1] It is not necessary that the values $\vec{\theta}_i$ and $\vec{\theta}_f$ are explicitly available to the learning algorithm. Here they are only listed as intermediate values for the derivation of the learning method and will later be replaced by expressions that will make it possible to use exclusively $\mathbf{v}_i$ and $\mathbf{v}_f$ provided by the cameras.

By this method one obtains an improved value $\mathbf{A}^* = \mathbf{A_s} + \Delta\mathbf{A_s}$ for $\mathbf{A_s}$ by setting

$$\Delta\mathbf{A_s} = \delta \cdot \Big(\Delta\vec{\theta}^{\nu} - \mathbf{A_s}\Delta\mathbf{v}^{\nu}\Big)(\Delta\mathbf{v}^{\nu})^T. \tag{11.11}$$

Here $\Delta\vec{\theta}^{\nu} = \vec{\theta}_f^{\nu} - \vec{\theta}_i^{\nu}$, $\Delta\mathbf{v}^{\nu} = \mathbf{v}_f^{\nu} - \mathbf{v}_i^{\nu}$, and δ is the learning step width. As long as $\delta \ll 1/\|\Delta\mathbf{v}\|^2$ and a stationary probability distribution of the quantities $(\vec{\theta}_i^{\nu}, \mathbf{v}_i^{\nu})$, $(\vec{\theta}_f^{\nu}, \mathbf{v}_f^{\nu})$ exists, then a sufficient number of steps (11.11) approximates a descent along the direction

$$\sum_{\nu}\Big(\Delta\vec{\theta}^{\nu} - \mathbf{A_s}\Delta\mathbf{v}^{\nu}\Big)(\Delta\mathbf{v}^{\nu})^T = -\frac{dE(\mathbf{A_s})}{d\mathbf{A_s}} \tag{11.12}$$

where $E(\mathbf{A_s})$ is the error function (11.10) to be minimized. Obviously, the Widrow-Hoff method leads to a minimization of $E(\mathbf{A_s})$ by realising a gradient descent of the mean square error $E(\mathbf{A_s})$ "on average." Although an individual step (11.11) may even increase $E(\mathbf{A_s})$, many adaptation steps (11.11) lead to an decrease of the error $E(\mathbf{A_s})$.
In applying (11.3), the equation determining the corrective movement, one can eliminate the explicit input of angle positions in learning step (11.11) and, thereby, employ only values provided by the cameras, namely the image coordinates $\mathbf{u}$, $\mathbf{v}_i$, and $\mathbf{v}_f$ of the target and the actual end effector positions. With $\Delta\vec{\theta}^{\nu} = \mathbf{A_s}(\mathbf{u}^{\nu} - \mathbf{v}_i)$ from (11.3) the adaptation step (11.11) can be written

$$\mathbf{A}^* = \mathbf{A}_{\mathbf{s}}^{old} + \delta \cdot \mathbf{A}_{\mathbf{s}}^{old}\Big(\mathbf{u}^{\nu} - \mathbf{v}_f^{\nu}\Big)(\Delta\mathbf{v}^{\nu})^T. \tag{11.13}$$

Since, in the course of learning, we approximate the Jacobian matrices with increasing accuracy, the learning step (11.9) for the expansion terms of 0-th order will deliver better and better values $\vec{\theta}^*$. Learning step (11.9) can also be interpreted as a gradient descent on a quadratic error function. In this case the quadratic error function is given by

$$E(\vec{\theta}_{\mathbf{s}}) = \frac{1}{2}\sum_{\nu}\Big(\vec{\theta}_i^{\nu} - \vec{\theta}_{\mathbf{s}} - \mathbf{A}_{\mathbf{s}}^0\Big(\mathbf{v}_i^{\nu} - \mathbf{w_s}\Big)\Big)^2. \tag{11.14}$$

Taking the derivative with respect to $\vec{\theta}_{\mathbf{s}}$ and employing Eq.(11.2) yields

$$-\frac{dE(\vec{\theta}_{\mathbf{s}})}{d\vec{\theta}_{\mathbf{s}}} = \sum_{\nu}\Big(\mathbf{A_s}\Big(\mathbf{u}^{\nu} - \mathbf{w_s}\Big) - \mathbf{A}_{\mathbf{s}}^0\Big(\mathbf{v}_i^{\nu} - \mathbf{w_s}\Big)\Big). \tag{11.15}$$

Since $\mathbf{A}_\mathbf{s}^0$ is initially unknown, we replace $\mathbf{A}_\mathbf{s}^0$ by the best available estimate, namely $\mathbf{A}_\mathbf{s}$. The error caused by this substitution is reduced by the improvement of $\mathbf{A}_\mathbf{s}$ with every trial movement and vanishes at the end of the learning process.[2] As for the Jacobian matrices, the adaptation step

$$\vec{\theta}^{\,*} = \vec{\theta}_\mathbf{s}^{\,old} + \delta \cdot \mathbf{A}_\mathbf{s}^{old}\Big(\mathbf{u}^\nu - \mathbf{v}_i^\nu\Big) \tag{11.16}$$

leads to a gradient descent "on average" on the function (11.14).
When this learning step is compared to Eq. (11.8) (where the index ν is again omitted), we recognize that both expressions become equivalent when $\mathbf{A}_\mathbf{s}^{old} = \mathbf{A}_\mathbf{s}^0$ and the step size is $\delta = 1$. Therefore, if the Jacobian matrices are learned correctly, we obtain through Eq. (11.16) also a correct new estimate $\vec{\theta}^{\,*}$ for the adjustment of the expansion terms of 0-th order, as long as we choose for the step size its optimal value $\delta = 1$.

11.4 Simulation Results

Having described the neural network algorithm for training the robot, we now present the results of a simulation involving this algorithm. The lattice which represents the work space consists of 7×12×4 neural units. As in previous simulations, the values of the parameters ϵ, ϵ', σ, and σ' decrease with the number of performed learning steps. For all four parameters we chose a time dependence of $x_i(x_f/x_i)^{t/t_{max}}$ with $t_{max} = 10,000$. For the initial and final values, x_i and x_f, we chose $\epsilon_i = 1$, $\epsilon_f = 0.005$, $\epsilon'_i = 0.9$, $\epsilon'_f = 0.5$, $\sigma_i = 3$, $\sigma_f = 0.1$, $\sigma'_i = 2$ and $\sigma'_f = 0.05$. The learning step widths δ_1 and δ_2 were set to their optimal values $\delta_1 = 1$ and $\delta_2 = 1/\|\Delta\mathbf{v}\|^2$.
The three arm segments of the robot arm simulated have a length of 0.13, 0.31 and 0.33 units, respectively, starting with the segment at the base of the robot. In the same units, the work space is a rectangular cube with $0.1 < x < 0.5, -0.35 < y < 0.35, 0 < z < 0.23$, where the x- and y-axes lie in the horizontal plane with the x-axis along the short edge. The aperture of camera 1 is located at $(x, y, z) = (0.7,\ 0,\ 0.12)$ and points towards the coordinate $(0.15,\ 0,\ 0)$; the aperture of camera 2 is located at $(0.3,\ 1,\ 0.25)$ and points towards $(0.3\ ,0\ ,0.2)$. Both of the two cameras have a focal length of 0.05 units.

[2] Due to this error, the learning step does not point exactly in the direction of the negative gradient.

The Kohonen net in the four-dimensional space cannot, of course, be directly displayed. In lieu of a direct display we show a projection of the network nodes from the four-dimensional input space onto the image planes of cameras 1 and 2. Each lattice point appears in the center of its receptive field on the image plane of cameras 1 and 2. If $\mathbf{w_r} = (\vec{w}_{\mathbf{r}1}, \vec{w}_{\mathbf{r}2})$ is the four-dimensional spatial vector of the lattice point $\mathbf{r}$, then we depict $\mathbf{r}$ at $\vec{w}_{\mathbf{r}1}$ on the image plane of camera 1 and at $\vec{w}_{\mathbf{r}2}$ on the image plane of camera 2. In this way the initial state of the net is shown in the two top frames of Fig. 11.4. The distribution of the network nodes was generated by assigning, on the image plane of camera 1, random values to the coordinate pairs $\vec{w}_{\mathbf{r}1}$ from a homogeneous probability distribution. The coordinate pairs $\vec{w}_{\mathbf{r}2}$ on the image plane of camera 2 were initialized accordingly. Therefore, we see in the top two frames of Fig. 11.4 a homogeneous distribution of the 336 lattice points. This implies that the initial distribution of the discretization points $\mathbf{w_r}$ in the four-dimensional space U is homogeneous, as well.

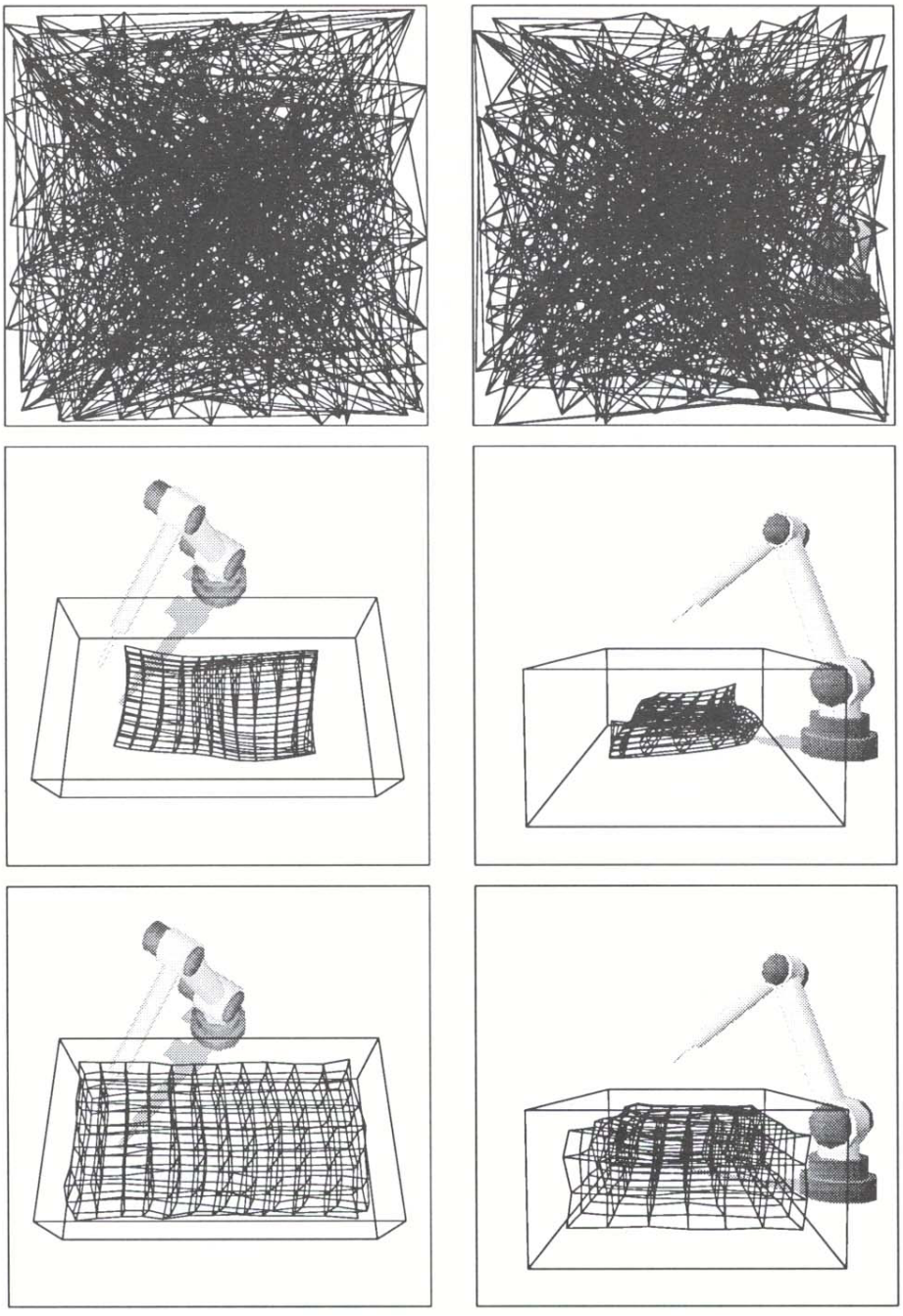

Abb. 11.4: Configuration of the neural lattice initially (top), after 2000 (middle), and after 6000 learning steps (bottom). The left column presents the image plane of camera 1, showing $\mathbf{w_{r1}}$ for all lattice nodes; the right column shows the image plane of camera 2, showing $\mathbf{w_{r2}}$

The two middle frames of Fig. 11.4 show the Kohonen net after 2000 learning steps. The work space and the robot arm have been displayed from the view of the corresponding camera. In Fig. 11.1 camera 1 is opposite to the robot arm, and camera 2 is to the right of the robot arm. Accordingly, we see the scene on the image planes of the cameras. In the top two frames of Fig. 11.4 no details can be seen yet because the connections between the elements of the Kohonen net completely cover the image planes. That the lattice, after 2000 learning steps, seems to "float" within the three-dimensional confines of the work space demonstrates that the Kohonen net represents already only the relevant subspace, however, only the central part of it.
The bottom frames of Fig. 11.4 show the Kohonen net after 6000 learning steps, a stage of the learning process where the positioning error has already reached its minimum value. The receptive fields of the network nodes have developed in a way that all possible target positions in the work space are equally represented. This gives rise to a "distortion" of the net to an extent that is determined by the affine projection of the work space onto the corresponding camera image plane and which in turn depends on the location of the camera. This can be seen particularly well in the image plane of camera 2 as shown in the right column of Fig. 11.4. For an homogeneous representation of the input signals, the network nodes on the image plane of camera 2 that are responsible for the back region of the work space must lie significantly more densely than the network nodes that represent target positions in the front part of the work space. By a proper distribution of its receptive fields according to the input stimulus density, the Kohonen algorithm yields this required "distortion" automatically.
To depict the learning of the output values, we again display frames from cameras 1 and 2 at different phases of the training. The two top frames of Fig. 11.5 show the initial state of the terms $\vec{\theta}_{\mathbf{r}}$ which provide the zero order contribution to the linear Taylor expansion (11.2) to be learned by each neural unit. The corresponding position of the end effector, after setting the joint angles $\vec{\theta}_{\mathbf{r}}$, is marked by a cross in each camera's visual field. These end effector positions are obtained by aiming at the target points $\mathbf{u} = (\vec{w}_{\mathbf{r}1}, \vec{w}_{\mathbf{r}2})$, for which the first order terms $\mathbf{A}_{\mathbf{r}}(\mathbf{u} - \mathbf{w}_{\mathbf{r}})$ in (11.2) vanish. At the end of the training, the camera coordinates of the end effector corresponding to $\vec{\theta}_{\mathbf{r}}$ should coincide with $\vec{w}_{\mathbf{r}1}$ and $\vec{w}_{\mathbf{r}2}$ of the lattice node $\mathbf{r}$. The deviations of the end effector positions from $\vec{w}_{\mathbf{r}1}$ and $\vec{w}_{\mathbf{r}2}$ indicate the residual error of the output values $\vec{\theta}_{\mathbf{r}}$. These errors are depicted in Fig. 11.5 in the visual field of

each camera by a corresponding line segment. The initial values of $\vec{\theta}_{\mathbf{r}}$ were chosen randomly, yet with the constraint that the end effector remains in front of the robot.

The development of the Jacobian matrices $\mathbf{A}_{\mathbf{r}}$ is illustrated in Fig. 11.6. For the purpose of illustration we assume that the robot arm is asked to perform small test movements parallel to the three edges of the work space. The desired movements if carried out correctly form a rectangular tripod. The starting positions for these test movements are the end effector positions that correspond to the joint angles $\vec{\theta}_{\mathbf{r}}$. Through the cameras we observe the test movements actually performed. Initially, because of the random assignment of the Jacobian matrices $\mathbf{A}_{\mathbf{r}}$, the test movements show no similarity whatsoever to the desired rectangular tripods. The initial values of the matrices $\mathbf{A}_{\mathbf{r}}$ were all chosen in the same way, namely, $A_{\mathbf{r}}^{ij} = \eta$ for all lattice points $\mathbf{r}$ where η is a random variable that is homogeneously distributed in the interval $[-10, 10]$.

The middle frames of Fig. 11.5 and Fig. 11.6 present the test of the 0-th order terms $\vec{\theta}_{\mathbf{r}}$ and of the Jacobian matrices $\mathbf{A}_{\mathbf{r}}$ after 2000 learning steps. At this stage the end effector positions resulting from the joint angles $\vec{\theta}_{\mathbf{r}}$ all lie within the work space (Fig. 11.5). The test movements for $\mathbf{A}_{\mathbf{r}}$ (Fig 11.6) already look approximately like rectangular tripods, except that the amplitudes of the movements are still too small. The bottom frames of Figs. 11.5 and 11.6 show the result after 6000 training instances, a stage where the positioning error has already reached its minimum; the output values $\vec{\theta}_{\mathbf{r}}$ and $\mathbf{A}_{\mathbf{r}}$ have been optimized. As desired, the end effector positions resulting from $\vec{\theta}_{\mathbf{r}}$, indicated by cross marks, now coincide with the image locations $\vec{w}_{\mathbf{r}1}$ and $\vec{w}_{\mathbf{r}2}$ of the network nodes, and the test movements for testing the Jacobian matrices are also performed as desired. Only near the base of the robot do test movements deviate slightly from the shape of a perfect rectangular tripod, an effect caused by a singularity in the transformation $\vec{\theta}(\mathbf{u})$. Because the Jacobian matrices $\mathbf{A}_{\mathbf{r}}^0$ represent the derivative $\vec{\theta}(\mathbf{u})$ at the locations $\mathbf{w}_{\mathbf{r}}$, some elements of $\mathbf{A}_{\mathbf{r}}$ must take on very large values. Therefore, a more precise adaptation requires an unacceptably high number of learning steps.

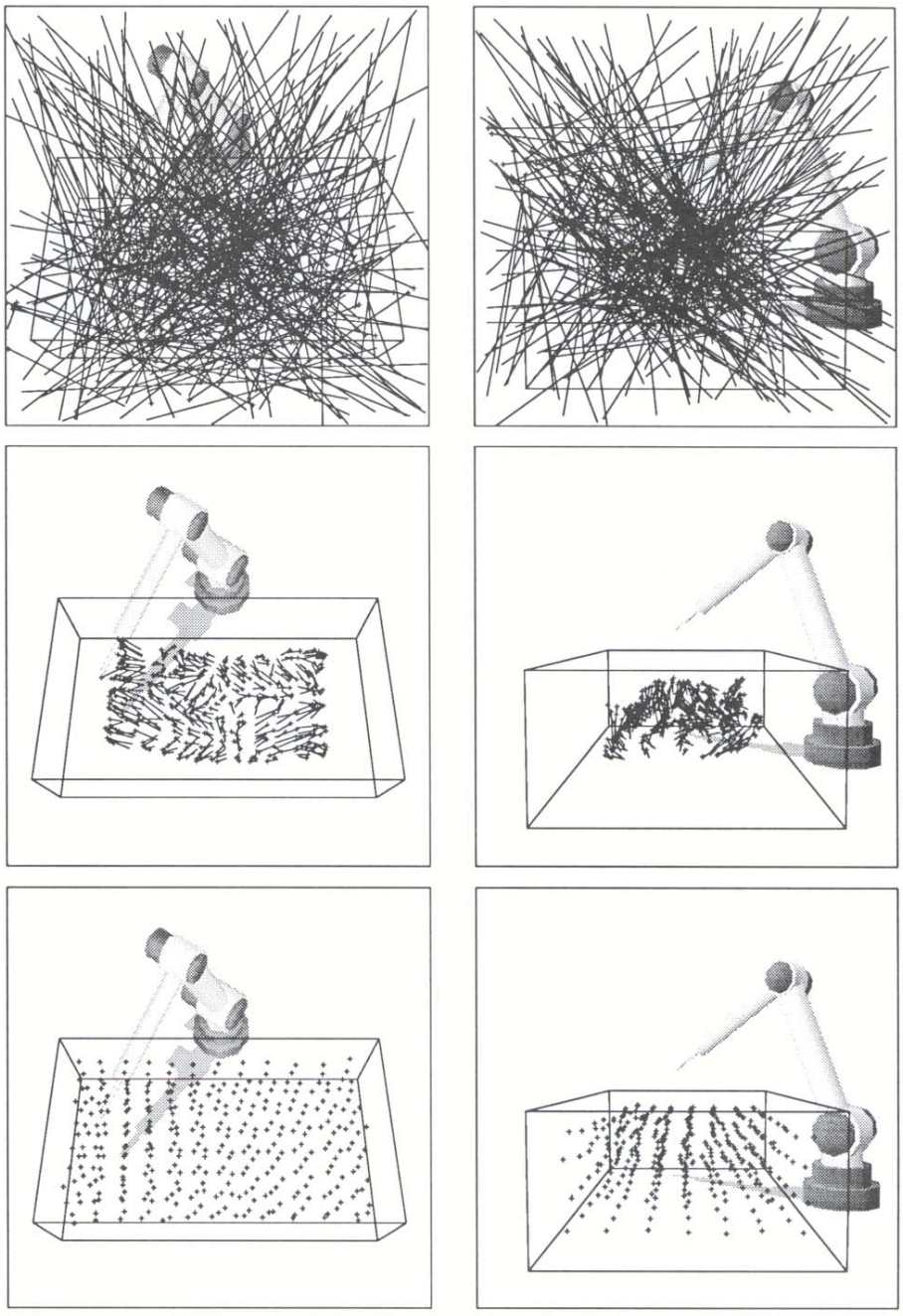

Abb. 11.5: The end effector positions corresponding to $\vec{\theta}_{\mathbf{r}}$ represented by cross marks. The deviation from the desired locations $\mathbf{u} = (\vec{w}_{\mathbf{r}1}, \vec{w}_{\mathbf{r}1})$ is displayed by lines appended to the cross marks. Shown are the states at the beginning (top), after 2000 (middle), and after 6000 learning steps (bottom). Left (right) column: view through the image plane of camera 1 (camera 2).

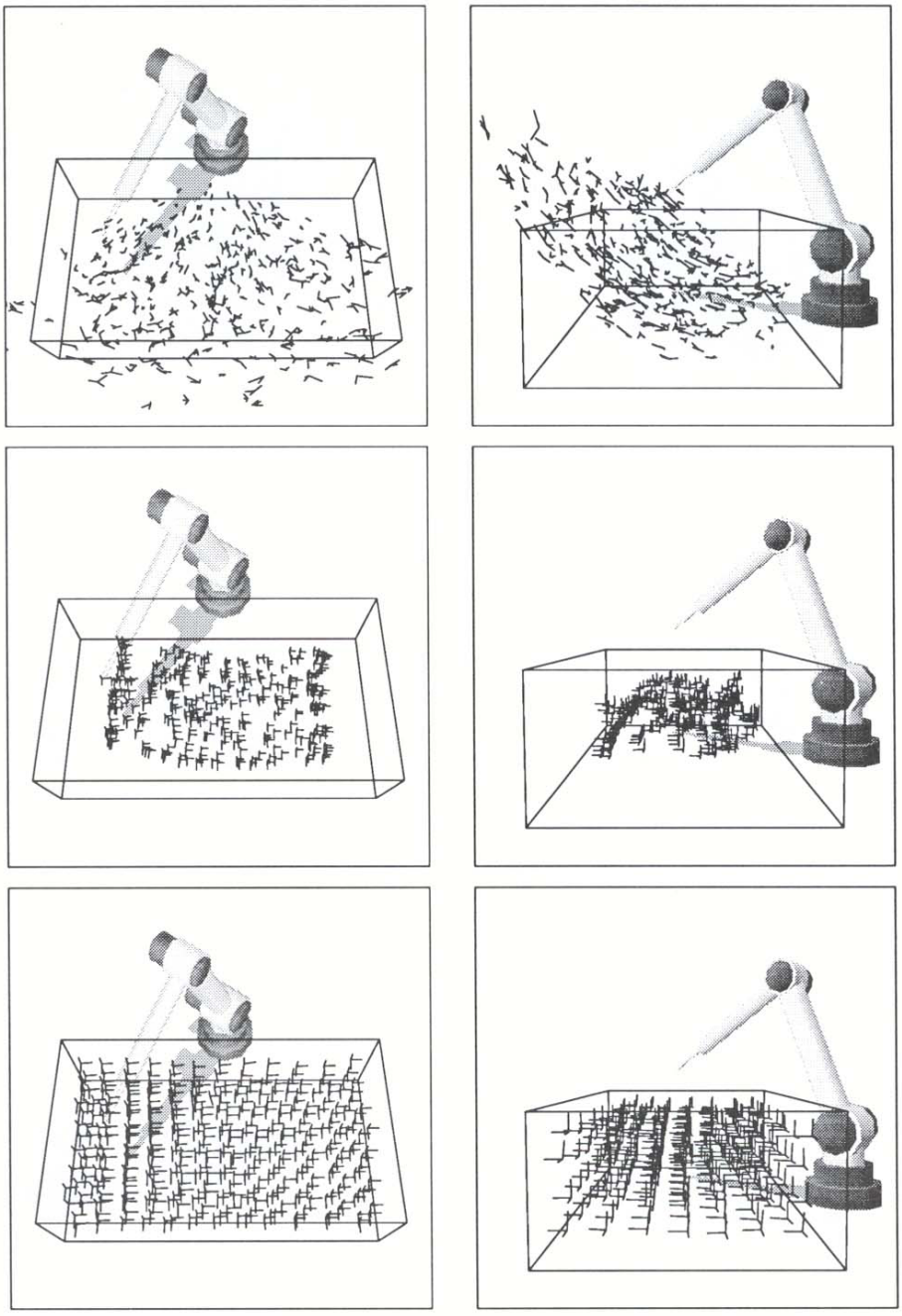

Abb. 11.6: Training of the Jacobian matrices $\mathbf{A_r}$. Displayed are the result of small test movements parallel to the edges of the work space, at the beginning, after 2000, and after 6000 learning steps. Left (right) column: view through the image plane of camera 1 (camera 2).

To demonstrate the success of the learning process we show in Fig. 11.7 the dependence of the mean positioning error on the number of trial movements. The mean positioning error at different instances is determined by an "intermediate test" after every 100 additional learning steps. During each intermediate test the robot system is presented with 1000 targets which are chosen at random within the work space. The arithmetic average of the end effector deviations from the targets provides the mean positioning error. During the intermediate tests the neural network remains untrained.

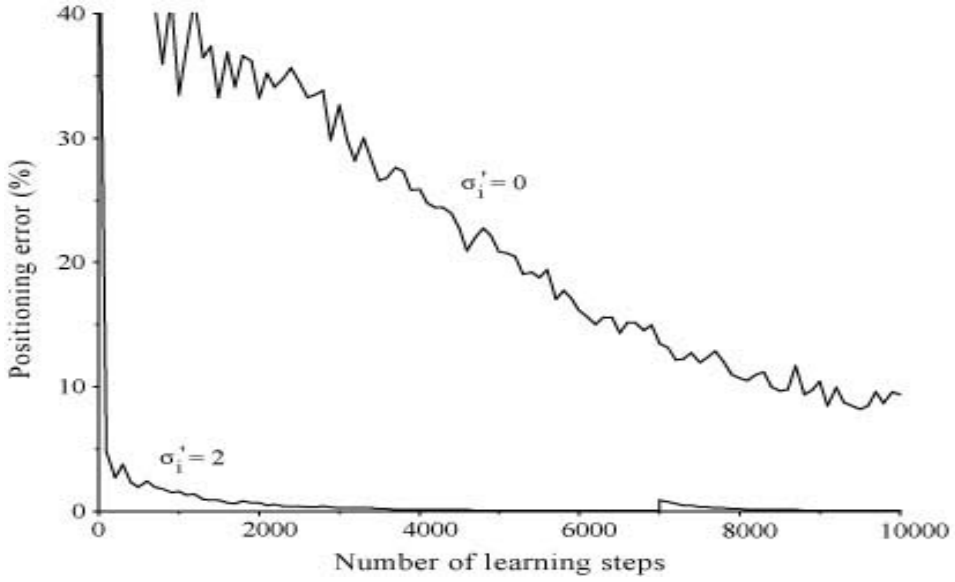

Abb. 11.7: The average deviation of the end effector from the target versus the number of training steps. σ' denotes the initial range of the cooperation between neighbors. Without cooperation between neighbors ($\sigma_i' = 0$) during the learning of the output values, training progress is slow and the system does not reach an error-free final state. With cooperation between neighbors ($\sigma_i' = 2$) the desired learning of an essentially error-free final state is achieved rapidly. The residual error after 6000 learning steps is only 0.0004 length units—about 0.06% of the length of the work space. After 7000 learning steps the third arm segment of the robot was extended by 0.05 length units, about 10% of the size of the robot arm. The resulting positioning error, which was initially 0.006 length units or about 1%, decays with more adaptation steps until the previous minimum value is regained.

The cooperation between neighboring neural units as described by (11.5) plays a decisive role in achieving a fast and precise visuomotor control. To demonstrate this we show in Fig. 11.7 the positioning error from a simulation without cooperation between neighbors ($\sigma_i' = \sigma_f' = 0$ for the neighborhood function $h'_{\mathbf{rs}}$ employed in (11.5)). The resultung error shows that the system no longer achieves the learning goal. The positioning error remains noticeable even after 10,000 trial movements. In contrast, cooperation between neighbors (initial range of cooperation of $\sigma_i' = 2.0$) induces the error to decay rapidly to a small, residual error. After 100 trial movements the positioning error, in case of cooperation, has decayed to 0.034 length units, *i.e.*, approximately to about 5% of the length of the work space. After about 6000 learning steps the positioning error has stabilized at a value of 0.0004 length units which corresponds to about 0.06% of the length of the work space. If the work space were one meter long, the robot would be able to move to target locations within a few tenths of a millimeter.

To demonstrate the adaptative capability of the neural network algorithm, we extended the last arm segment to which the end effector is connected by 0.05 length units. This corresponds to a change of about 10% of the total length of the robot arm. Immediately after this modification, the positioning error increases since the neural net needs a few adaptation steps to be able to adjust to this new configuration. It is remarkable that immediately after the modification the positioning error is about 1%—smaller by a factor of ten than one would expect from the size of the modification. This is due to the feedback inherent in the positioning process: the second step in the positioning of the end effector, the corrective movement $\Delta\vec{\theta} = \mathbf{A_r}(\mathbf{u} - \mathbf{v}_i)$, corrects the deviation of the intermediate end effector position $\mathbf{v}_i$ from the targeted position $\mathbf{u}$, which has been increased by the modification of the robot arm. Extension of the last arm segment changes, for the most part, the target values of the zeroth order expansion terms $\vec{\theta}_{\mathbf{r}}$. In contrast to that, the Jacobian matrices $\mathbf{A_r}$ are only affected slightly by the arm extension and, therefore, are still suitable for the execution of the corrective movement and for the correction of the error resulting from the wrong $\vec{\theta}_{\mathbf{r}}$. Figure 11.7 shows that after a few learning steps the neural net has completely adapted to the arm extension, and the positioning error has decreased to its previous small value.

The capability of the algorithm to adapt immediately to small changes in the robot arm adds significantly to its flexibility. For example, robots trained and controlled by the suggested algorithm could be equipped with different

tools without the need for an entirely new course of training.
In the following section we will demonstrate the high degree of flexibility of the described learning algorithm in another respect. It turns out that the neural network can control a robot arm with more joints than necessary to position the end effector to any point in the three-dimensional workspace without additional modifications in the learning method. We will show how the neural network algorithm can handle such robots with "redundant degrees of freedom" which imply the difficulty that no unique relationship exists between joint angles and end effector positions.

11.5 Control of a Robot Arm with Redundant Degrees of Freedom

The triple-jointed robot that we discussed in previous sections could reach any target location within the work space by a unique set of joint angles.[3] Mathematically speaking, this implies that there existed a one-to-one mapping between target positions and sets of joint angles.
Nonetheless, most organisms capable of movement possess extra degrees of freedom. The human arm, for example, has four degrees of freedom: three in the shoulder joint and one in the elbow. It is indeed possible for humans to reach an object by many different arm configurations. With more than three degrees of freedom the set of joint angles is not uniquely determined by the location of the target, but rather there is a whole range of different sets of angles which will reach the target. From this range, a single set must be chosen. Such a problem is called an *ill posed* or *under-determined* problem since the constraints that must be fulfilled do not uniquely determine the solution (Bernstein 1967; Saltzman 1979; Jordan and Rosenbaum 1988).
The redundant degree of freedom in the human arm is, of course, not superfluous; it is particularly convenient when certain configurations are not possible due to additional constraints, *e.g.*, due to obstacles or when unique approaches are compulsory. A huge palette of arm configurations offers the chance to find one that works well.
In the following, however, we will not consider the implications of such constraints. We will stick to the task of positioning the end effector in a work

[3] Actually there exist two sets of joint angles which lead to a given target location. They correspond to a "convex" and a "concave" configuration of the arm. By prohibiting, *e.g.*, the concave configurations, the required set of angles becomes unique.

space free of obstacles. We want to study the learning performance of the algorithm of the previous sections as it is applied to an arm with more than three degrees of freedom. Which arm configurations will the algorithm learn when it can select from many possibilities for each target?

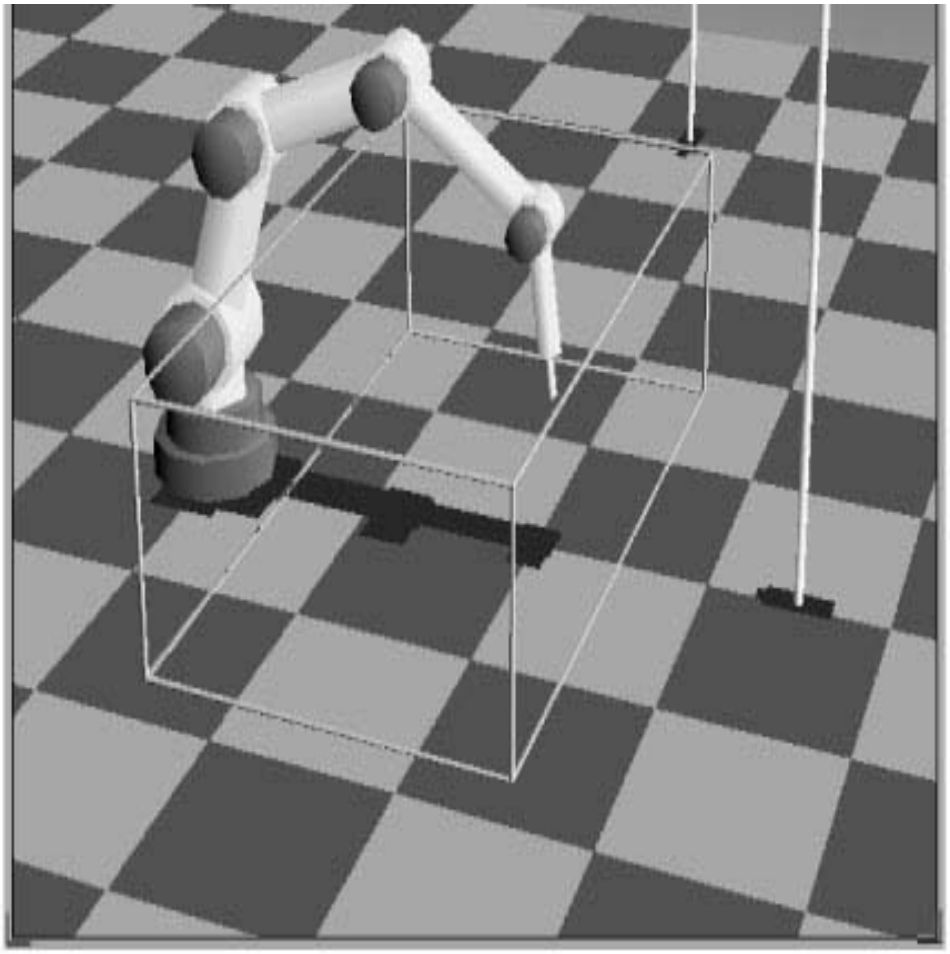

Abb. 11.8: Simulated robot with redundant degrees of freedom. The arm has a total of five joints: one joint permitting rotation around the vertical axis and four joints about which the arm can move in the vertical plane.

Common methods for the control of an arm with redundant degrees of freedom eliminate the under-determination of the control problem by selecting

that joint angle configuration that minimizes a reward function in addition to reaching the target point (Kirk 1970; Hogan 1984; Nelson 1983). Psychophysical experiments show that humans also prefer arm configurations that minimize certain "costs." It has been shown (Cruse et. al. 1990) that humans prefer arm configurations using middle-range angles, *i.e.*, fully extended or tightly contracted angles are generally avoided. Such a tendency can be modelled in an algorithm by selecting from all possible arm configurations the one that minimizes a suitable reward function, such as,

$$E(\vec{\theta}) = \sum_{i=1}^{L} \left(\theta_i - \theta_i^{(0)}\right)^2 . \tag{11.17}$$

Here $\theta_i^{(0)}$ is a middle-ranged value for joint i. Joint angle configurations that deviate from $\theta_i^{(0)}$ increase the "costs" and are therefore less desireable.
Another form of the reward function is obtained if the arm is supposed to be as "lazy" as possible while moving, *i.e.*, the change of the joint angles of the arm should be as small as possible. This is a sensible, real-life requirement; it reduces both the wear and tear and the energy consumption.
If we denote the difference between two points that are neighbors, either in space or in camera coordinates, by $\Delta\mathbf{u}$, then the norm $\|\Delta\vec{\theta}\|$ of the difference between the joint angles which correspond to these points should be as small as possible on average. Mathematically, this constraint can be written as

$$\left\langle \sum_{i,j} \left(\frac{\Delta\theta_i}{\Delta u_j}\right)^2 \right\rangle_{\Delta\mathbf{u}} = \text{Min.} \tag{11.18}$$

Here $\Delta\theta_i$ denotes the change of the i-th joint angle and Δu_j denotes the difference in the j-th component of the space or camera coordinates. If we assume that the distribution of the occurring movements $\Delta\mathbf{u}$ which is averaged over in Eq. (11.18) is isotropic—which implies $\langle \Delta\mathbf{u}\Delta\mathbf{u}^T \rangle = \gamma\mathbf{1}$ where γ is a scalar and $\mathbf{1}$ is the identity matrix—then it can be shown that the constraint to minimize the reward function (11.18) is equivalent to the constraint that, for the inversion of the transformation $\mathbf{u}(\vec{\theta})$, the particular inverse $\vec{\theta}(\mathbf{u})$ must be selected from the whole range of possibilities for which the norm $\|\mathbf{A}\| = \sqrt{tr\mathbf{A}\mathbf{A}^T}$ of the Jacobian

$$\mathbf{A}(\mathbf{u}) = \frac{\partial\vec{\theta}(\mathbf{u})}{\partial\mathbf{u}} \tag{11.19}$$

is minimal at each location $\mathbf{u}$. It is natural then to look for the inverse that fulfills this constraint.
The Jacobian matrices $\mathbf{A}(\mathbf{u})$ of all the possible inverse transformations $\vec{\theta}(\mathbf{u})$ generate joint angle changes $\Delta\vec{\theta} = \mathbf{A}(\mathbf{u})\Delta\mathbf{u}$ for a given $\Delta\mathbf{u}$. If we denote by $\hat{\mathbf{B}}(\vec{\theta})$ the Jacobian matrices of the so-called "forward transformation" $\mathbf{u}(\vec{\theta})$, which is, in contrast to its inverse transformation $\vec{\theta}(\mathbf{u})$, uniquely determined and which is often easily obtained, then it holds for every pair of corresponding $\Delta\mathbf{u}$, $\Delta\vec{\theta}$,

$$\Delta\mathbf{u} = \hat{\mathbf{B}}(\vec{\theta})\Delta\vec{\theta}. \tag{11.20}$$

The Jacobian matrices $\mathbf{A}(\mathbf{u})$ of the inverse $\vec{\theta}(\mathbf{u})$ for which we are searching must obey the condition

$$\hat{\mathbf{B}}\Big(\vec{\theta}(\mathbf{u})\Big)\mathbf{A}(\mathbf{u}) \;\; = \;\; \mathbf{1}.$$

$\mathbf{A}$ cannot be determined from this equation by inversion of $\hat{\mathbf{B}}$ because in the case of redundant degrees of freedom, $\hat{\mathbf{B}}$ is rectangular and, therefore, not invertible. Here we face the same situation as in Chapter 3 where we had to determine the optimal memory matrix of an associative memory. There we obtained a solution with the help of the pseudo-inverse of $\hat{\mathbf{B}}$ (Albert 1972). From Chapter 3 we know that upon selecting the pseudo-inverse, the constraint to minimize the norm $\|\mathbf{A}\|$ is fulfilled simultaneously. Thus, the desired solution for $\mathbf{A}$ is

$$\mathbf{A} = \lim_{\alpha\to 0} \hat{\mathbf{B}}^T\Big(\hat{\mathbf{B}}\hat{\mathbf{B}}^T + \alpha\mathbf{1}\Big)^{-1}. \tag{11.21}$$

Of the many possible transformations that are inverse to the transformation $\mathbf{u}(\vec{\theta})$, there is one that generates movements with minimal changes in the joint angles. If that particular transformation is called for, then the inverse $\vec{\theta}(\mathbf{u})$ whose Jacobian matrices are given by the pseudo-inverse (11.21) of the Jacobian matrices of $\mathbf{u}(\vec{\theta})$ yields the solution.
In robotics the precise form of the reward function is often only of minor importance. The reward function often only serves to smooth the robot arm's movements. For two adjacent target points an algorithm could select two completely different arm configurations when there are redundant degrees of freedom. By employing a reward function one ensures that adjacent target points will be assigned to similar arm configurations.
The assignment of similar joint angles to adjacent target points is, in fact, one of the main features of our learning algorithm. In contrast to other common

methods, we do not have to minimize an explicitly formulated reward function. By the construction of a topographic map between input signal space and neural net it is made sure that adjacent target points always activate adjacent elements in the network. In addition, learning step (11.5) forces adjacent lattice nodes to adapt their output towards similar values. At the end of the learning phase the output values will vary smoothly from node to node. Both features bring about a continuous and smooth transformation from the input signal space of target points to the output space of joint angle sets. This transformation guarantees smooth and efficient movements of the arm.

11.6 Simulation Results

In Fig. 11.8 we see the robot system as simulated in the computer, now composed of five joints and, therefore, having two redundant degrees of freedom. As before the robot arm can pivot around its vertical axis, and the other joint axes are parallel to each other and parallel to the horizontal plane. The length of the arm segments were chosen as follows, beginning at the base of the robot: 0.13, 0.19, 0.19, 0.19, and 0.15 length units. The Kohonen net used in this simulation had the same size as the previous one, namely $7\times12\times4$ lattice nodes. The parameters ϵ, ϵ', σ and σ' as well as their time dependence, the location of the cameras, and the parameters describing the work space were adopted unchanged from the simulation of the triple-jointed robot arm. In connection with the five joints the vectors $\vec{\theta}_{\mathbf{r}}$ now have five components, and the matrices $\mathbf{A}_{\mathbf{r}}$ are now 5×4-dimensional.

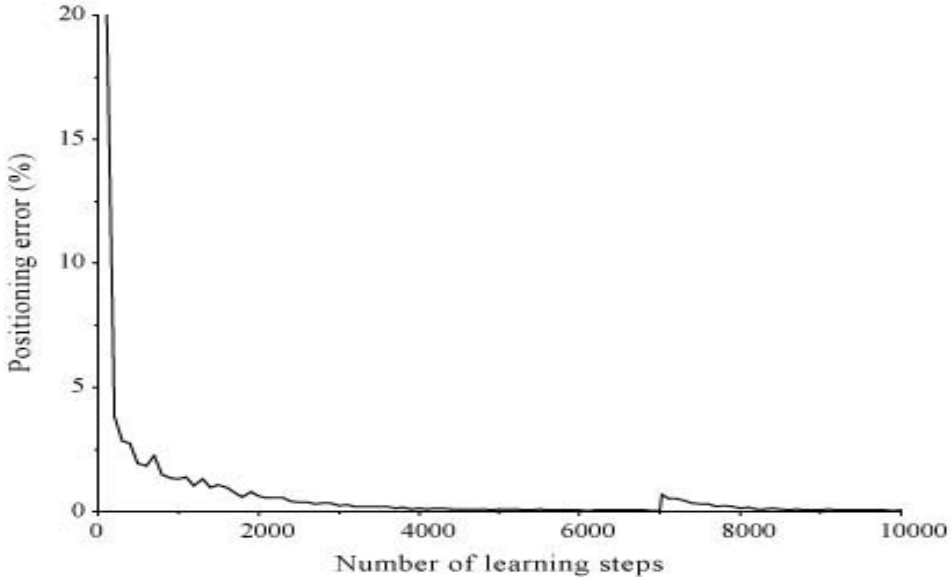

Abb. 11.9: The average deviation of the end effector from the target during training for a robot system with redundant degrees of freedom. As for the robot arm with only three joints the residual error after 6000 learning steps is 0.0004 length units, i.e., approximately 0.06% of the length of the work space. In order to test the adaptation capability we extended the last arm segment after 7000 learning steps by 0.05 length units. The resulting positioning error decays with additional adaptation steps until it has regained its previous value of 0.0004 after 2000 additional learning steps.

The graph in Fig. 11.9 shows the decay of the positioning error with the number of performed learning steps. As with the robot arm with only three degrees of freedom the error decays rapidly in the beginning and reaches, after only 200 learning steps, a value of 0.027 length units, which corresponds to about 4% of the length of the work space. It is noteworthy that the positioning error after 6000 learning steps reaches the same minimal value of 0.0004 length units as the triple-jointed robot, in spite of the increased number of degrees of freedom. After 7000 trial movements, we again extended the last arm segment by 10% of the length of the robot arm to test the adaptational capability of the system. As in the triple-jointed case, the positioning error initially increases but then decays again with additional adaptation steps until the previous value of 0.0004 learning steps is regained.

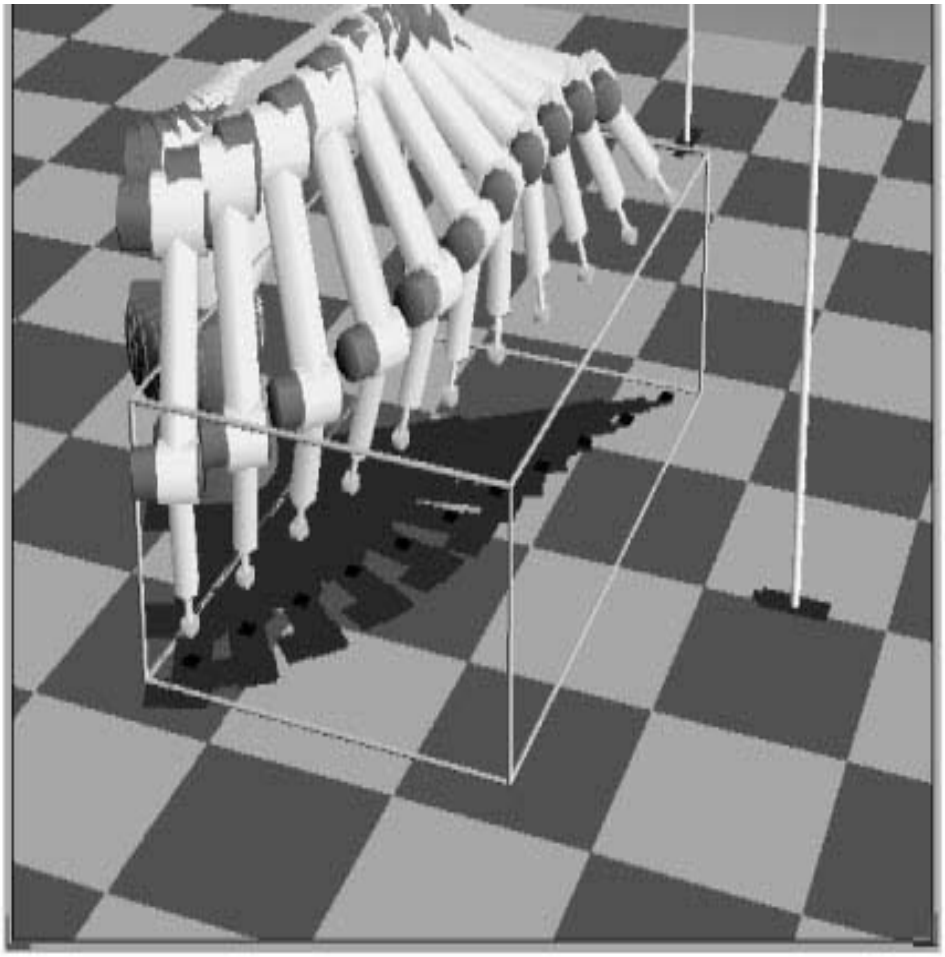

Abb. 11.10: Stroboscopic rendering of a movement of the robot arm. The robot arm passes along the diagonal of the work space with its end effector. Due to the redundancy of the robot arm each point along the trajectory can be reached by an infinite number of joint angle sets. The topology-preserving feature of the algorithm forces the use of joint angles that give rise to a smooth movement.

In order to test how well adjacent target points are reached by similar arm configurations, the robot is commanded to move along a trajectory of target points. In Fig. 11.10 the arm's movement in such a test is shown by monitoring the arm's position at different times. We recognize that the robot indeed

performs a smooth motion while moving its end effector along the diagonal of the work space. This demonstrates that the learning algorithm adapts to those arm configurations out of the many possible ones that lead to smooth changes in the joint angles as the arm moves. This is achieved without the explicit minimization of a reward function. The development of a topology--preserving mapping between the input signal space and the neuron lattice, and between the space of output values and the neuron lattice alone does the job (Martinetz et al. 1990b).

11.7 The Neural Network as a "Look-Up Table "

The presented network for the learning of visuomotor control of arm movements can adapt to arbitrary, continuous, nonlinear input-output relations. The learning algorithm belongs to the class of *"learning by doing"* methods. The approach of the algorithm really amounts to the generation of a *look-up table.* After presentation of an input value (target position) an entry $(\vec{\theta}_{\mathbf{s}}, \mathbf{A}_{\mathbf{s}})$ must be taken from a table (Kohonen net) that determines the joint angles that are needed to reach the target. The table entry, however, does not immediately deliver the required joint angles, which would correspond to the approximation of the transformation $\vec{\theta}(\mathbf{u})$ to be learned by a step function, but rather two expansion terms for the representation of $\vec{\theta}(\mathbf{u})$ up to linear order are made available. From the expansion terms the required joint angles can be determined to a much greater precision. Each table entry is responsible for only a small subregion of the input signal space; thus, an approximation of the input-output relation to be learned is obtained by a covering of the input signal space with locally valid linear mappings.
The use of Kohonen's algorithm offers the advantage of a flexible wiring between input values and the entries (network nodes) in the table (network), which depends on the input values which have already occurred. The Kohonen algorithm ensures that, in fact, all slots of the table are utilized, and by this the limited number of possible entries are optimally employed. Furthermore, the Kohonen algorithm distributes the entries in the table such that entries which are neighbors in the table are assigned to input values which are neighbors in the input space, based on a given metric. In our example we used the Euclidean metric in the camera coordinates. In order to represent the three-dimensional work space in this neighborhood-conserving

way, we employed a three-dimensional "table." In addition to the Kohonen algorithm which exclusively organizes the distribution of the entries of the table, we used an algorithm for the learning of suitable entry contents. Here we used an adaptation step following the Widrow-Hoff rule. An additional cooperation between neighboring entries of the table supports the convergence of the learning algorithm, because, as a consequence of the Kohonen algorithm, neighboring table entries must take on similar contents. It even turns out that without the transfer of each learning success to neighboring entries, the table (neural net) does not converge towards the desired state.
The visuomotor control task with its input-output relation $\vec{\theta}(\mathbf{u})$ is one example among many other control tasks which can be learned by the neural network described. In a real-life implementation there is an opportunity to further abstract the notion of input and output values. Why should the image point coordinates be used explicitly as input values and why should the output values provide explicitly the angles to be set? The presented method adapts arbitrary continuous nonlinear input-output relations. Therefore, a sufficient prerequisite for the application of the learning method is the use of a continuous coding of the input and output signals. Instead of the image point coordinates that we have used so far, the voltage values provided from the cameras could be directly used as input signals for the neural net, without requiring the precise knowledge about the relation between object positions and voltage values. Correspondingly, we would not have to take care of the precise transformation from voltage impulses into rotation movements of the joint motors, or of the amplifiers, filters etc. that might be installed between the input and the output. In case of an "antropomorphic" arm, output values that describe muscle contractions could be delivered instead of joint angles. The algorithm would autonomously adapt the complicated nonlinear transformation from muscle contractions to joint angles. It is only necessary that the response of the robot arm to target points is continuous and reproducible. The coding of the input-output relation can remain unknown and also the inner workings of the robot system may be considered as a "black box." The neural network will adapt to all unknown specifications with the help of the learning algorithm and will achieve its learning goal without help from outside.
With this we conclude this chapter about the learning of one of the basic tasks in robotics, namely end effector positioning. After the robot has learned to reach objects at arbitrary locations in the work space, the next step will be

to learn simple manipulations. For these manipulations in most of the cases the grasping of objects will be necessary. The grasping of objects is the next basic task that a robot must accomplish after end effector positioning, which is the reason that we consider this problem more closely in the following chapter.