Message inbox:478 - Read From: Carme Torras ¡ctorras@iri.upc.es¿ To: ¡vruiz@iri.upc.es¿ Subject: Articulo (include)

—————————— Cut here ——————————

Self-calibration of a space robot

Vicente Ruiz de Angulo and Carme Torras

The authors are with the Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Edifici NEXUS, Gran Capità 2-4, 08034-Barcelona, Spain. E-mail: vruiz@iri.upc.es, ctorras@iri.upc.es. A preliminary version of this paper appears in the Proceedings of the Intl. Conf. on Artificial Neural Networks (ICANN'96).

Abstract

We present a neural network method to recalibrate automatically a commercial robot after undergoing wear or damage, which works on top of the nominal inverse kinematics embedded in its controller. Our starting point has been the work of Ritter et al. [?] on the use extended self-organizing maps to learn the whole inverse kinematics mapping from scratch. Besides adapting their approach to learning only the deviations from the nominal kinematics, we have introduced several modifications to improve the cooperation between neurons. These modifications not only speed up learning by 2 orders of magnitude, but also produce some desirable side effects, like parameter stability. After extensive experimentation through simulation, the recalibration system has been installed in the REIS robot included in the space-station mock-up at Daimler-Benz Aerospace. Tests performed in this set-up have been constrained by the need to preserve robot integrity, but the results have been concordant with those predicted through simulation.

Keywords

Inverse kinematics, robot manipulator, automatic recalibration, self-organizing maps.

I. Introduction

The recalibration of robots installed in unmanned space stations through teleoperation from earth is a very time-consuming task due to communication delays. Within the project CONNY[1], Daimler-Benz Aerospace S.A. (D.A.S.A.) proposed an application of maintenance of electronic equipment that required the automatic recalibration of a 6-dof robot in-situ after wear had occurred. We present here the solution implemented in the REIS robot included in the space-station mock-up installed in D.A.S.A.'s R&D laboratory, in Bremen, Germany. Robot recalibration in this context means adapting the inverse kinematics model embedded in the robot controller to changes in either the environment or the own geometry of the robot, due to the launching stress, repeated usage or damage.

The most influential work on unsupervised learning of inverse kinematics with neural networks is that of Ritter et al. [?], [?], [?], [?]. They use hierarchical self-organizing maps to learn the whole kinematic mapping from scratch, the efficiency and originality of their approach relying on the cooperation of the learning units. One of the advantages of their approach is that it permits successive refinements of the positioning movement

[1]Project "Robot Control based on Neural Network Systems, within the ESPRIT III Program of the European Union. The remaining partners, besides our institute, were: Daimler-Benz Aerospace (D), THOMSON CSF (F), Mimetics S.A. (F), Framentec-Cognitech (F), CRAM-AID (I) and University College of London (UK).

by employing visual feedback, thanks to the direct representation of the Jacobian of the mapping in the model. We have modified this model in several ways to suit a more practical setting.

Current commercial robot arms are manufacturated with an execution controller that includes accurate models of their inverse kinematics in standard conditions. The user can only move the robot to different positions through commands to this control module. It acts as an unavoidable interface for the user, who must provide control commands specifying desired positions.

Here we present a method to recalibrate automatically the robot when it undergoes wear or damage, without replacing the original interface. The idea is to learn the implicit mapping from robot poses to the commands supplied to the controller previous to the attainment of those poses. This is an inverse mapping of the same kind as the one learned in the primitive application. However, there is a difference that induced one of the main changes in the algorithm: While the inverse kinematics mapping was learned from scratch, our inverse function is known to be initially the identity and only the distortions have to be learned.

On the other hand, we also focus on the improvement of several aspects of the cooperation among neurons, which give raise to a quicker and more principled algorithm.

II. Ritter et al.'s approach to learning inverse kinematics

Ritter et al.'s model, as applied to a 5-dof robot, consists of a 3D self-organizing map (SOM) whose nodes have associated a 2D (SOM) each. Learning makes the 3D net converge to a discrete representation of the workspace, while the 2D subnet represents the gripper orientation space.

When a given position $\mathbf{u_p}$ and orientation $\mathbf{u_o}$ are supplied as input, the subnet k with input weights $\mathbf{w}_k$ closest to $\mathbf{u_p}$ is selected and, within this subnet, the neuron l with input weights $\mathbf{w}_{kl}$ closest to $\mathbf{u_o}$ is chosen. The joint angles produced for this particular input are then obtained with the expression:

$$\theta' = \theta_{kl} + \mathbf{A}_{kl}((\mathbf{u_p}, \mathbf{u_o}) - (\mathbf{w}_k, \mathbf{w}_{kl})), \tag{1}$$

where θ_{kl} and $\mathbf{A}_{kl}$ are respectively the vector of joint angles and the 5×8 Jacobian matrix associated with the winning neuron kl.

A learning cycle consists of the following four steps:

1. First, the classical Kohonen rule is applied to all weights $\mathbf{w}_i$ and $\mathbf{w}_{ij}$.

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + c\ h_k(i)\ (\mathbf{u} - \mathbf{w}_k^{old}), \tag{2}$$

$$\mathbf{w}_{ij}^{new} = \mathbf{w}_{ij}^{old} + c\ h_{kl}(j)\ (\mathbf{u} - \mathbf{w}_{kl}^{old}). \tag{3}$$

2. By applying θ' to the real robot, the end-effector moves to pose $\mathbf{u}' = (\mathbf{u_p}', \mathbf{u_o}')$. The difference between this pose and the desired one $\mathbf{u} = (\mathbf{u_p}, \mathbf{u_o})$ constitutes an error signal that permits applying the LMS rule:

$$\theta^* = \theta_{kl} + \mathbf{A}_{kl}(\mathbf{u} - \mathbf{u}'). \tag{4}$$

3. By applying the correction increment $\mathbf{A}_{kl}(\mathbf{u}-\mathbf{u}')$ to the joints of the real robot, θ'' and a corresponding refined position $\mathbf{u}''$ are obtained. Now, the LMS rule can be applied to the Jacobian matrix by using $\Delta\theta = (\theta'' - \theta')$ as the error signal for $\Delta\mathbf{u} = (\mathbf{u}'' - \mathbf{u}')$:

$$\mathbf{A}^* = \mathbf{A}_{kl} + (\Delta\theta - \mathbf{A}_{kl}\Delta\mathbf{u})\frac{\Delta\mathbf{u}^T}{\|\Delta\mathbf{u}\|^2}. \tag{5}$$

4. Finally, the Kohonen rule is applied to the joint angles:

$$\theta_{ij}^{new} = \theta_{ij}^{old} + c'\ g_k(i)\ g_{kl}(j)\ (\theta^* - \theta_{ij}), \tag{6}$$

and the Jacobian matrix:

$$\mathbf{A}_{ij}^{new} = \mathbf{A}_{ij}^{old} + c'\ g_k(i)\ g_{kl}(j)\ (\mathbf{A}^* - \mathbf{A}_{ij}), \tag{7}$$

where again c' is the learning rate and $g_k(.)$ and $g_{kl}(.)$ are Gaussian functions centered at k and kl, respectively, used to modulate the adaptation steps as a function of the distance to the winning neuron. The widths of the Gaussians decrease with time.

Note that essentially the algorithm generates a look-up table. However, a table entry, instead of delivering directly the required joint angles, provides a linear approximation of the function from which the angles can be determined with more precision. The hierarchical structure of the network serves the purpose of reducing the search time for the winning unit. While operating with the robot, if $\mathbf{u}''$ is not sufficiently accurate, the precision can be improved once again by correcting θ'' with $\mathbf{A}_{kl}(\mathbf{u} - \mathbf{u}'')$. This use of the visual feedback can be repeated a certain number of times to refine the movement.

III. A new application: robot recalibration

As explained in the introduction, we approach the calibration problem from an industrial point of view. We desire to automatically recalibrate any commercial robot already provided with an execution controller which cannot be substituted and, in the way, profit from the initial accuracy provided by it. In particular, we want to control the three position coordinates and the three orientation coordinates of a robot with six degrees of freedom. A simple option would be to learn the inverse mapping between camera coordinates and commands, but, as another module of the CONNY project faithfully provides us with the real pose coordinates (the technique used is similar to that in [?]), we choose a more advantageous strategy.

Our application entails learning the mapping f^{-1} from desired robot poses to appropriate pose commands which, when supplied to the controller, lead to the attainment of those desired poses. Figure 1 illustrates the mapping. An initial learning is not required for a brand-new robot, because for an intact arm, f^{-1} is known to be the identity. After some degradation, the learned mapping amounts to sending the robot to a fake pose in order for it to reach the desired one. This approach avoids also the problem, present in the original application, of having a multivalued inverse function, because the controller always chooses the same joint angles for a fixed command. Thus, f^{-1} is a bijective mapping. However, we still need the indirect style of Ritter et al. to learn the mapping, because we cannot get directly the command which produces a given pose, instead we can only determine what output corresponds to a given input. In other words, we are also learning an inverse function.

IV. Evaluation of Ritter et al.'s algorithm

Along this paper we will explain the difficulties we encountered in applying Ritter et al.'s algorithm to robot recalibration and how we solved them. First of all, taking into account the details of the new application, some simplifications to the original system can be made.

A. Adapting the algorithm to the new setting

The workspace shape is perfectly known and not open to change along time, unlike camera coordinates that are subject to physical distortions. Thus, we can directly initialize the centers of the cells to conform a regular grid covering the workspace, so that the quantization error is minimized. These centers do not need to move, if the workspace shape does not change and, thus, (2) is bypassed in all the experiments presented here. Note that, in this case, the search for the winning neuron in rectangular workspaces can be made much more efficient, by carrying it out independently for each position and orientation coordinate component. In this way, the hierarchical structure loses its sense and the algorithm can be simplified. However, in the conclusions we suggest that moving the units with a modified rule could still have beneficial effects and, for this reason, we adhere to the hierachical notation in what follows.

Another consideration is that, if the contribution of each dimension is weighed with the density of the units in that dimension, the euclidean distance can be made proportional to the lattice one. We used the euclidean distance hoping to obtain independence of the parameters associated to the neighborhood with respect to the number and distribution of units in the network and to the size of the workspace.

Next we summarize the adaptations of the original Ritter et al.'s algorithm to the modified setting described above. To keep the terminology introduced in Section 2, from now on $\mathbf{u}$ and θ will represent six-dimensional vectors denoting pose coordinates and commands, respectively. The modifications are as follows:

- The network is now asked to learn the new f^{-1} function from pose coordinates to controller commands.

- The weights $\mathbf{w}_k$ and $\mathbf{w}_{kl}$ are set initially to cover regularly the chosen workspace. They do not move along the learning.

- To represent the identity mapping, all the centers θ_{kl} of the cells are initialized to $(\mathbf{w}_k, \mathbf{w}_{kl})$. All the Jacobian matrices $\mathbf{A}_{kl}$ are initialized to the identity matrix.

- The euclidean distance instead of the lattice one is used to evaluate $g_k(.)$ and $g_{kl}(.)$.

B. Experimental results to be used as reference

Since evaluation on the real robot was necessarily restricted (see Section IX), we used simulation in the initial stages of development. We present now an extensive simulation study of the behavior of the system for a case more interesting and representative than those allowed on the real robot, i.e., one in which the geometry of the robot undergoes serious distortion. The robot had to maneuver in a workspace of $50 \times 60 \times 50$ cm with an orientation range of 40 degrees in each dimension. The length of three links were shortened by 1, 1 and 4 cm, respectively, while three joint encoders were shifted by 4, 3, and 4 degrees. This could result, for instance, from link bending and encoder wear. As a consequence, the initial mean average position and orientation error, when executing (1), was 8.3 cm and 4.7 degrees, respectively.

We carried out an optimization of all relevant parameters of the algorithm as carefully as allowed by our computational resources. In the optimization experiments, we decreased the main parameters of the algorithm (the learning rate c', the standard deviation of the neighborhood in position space σ, and that in orientation space σ') with the time dependence usually chosen by Ritter et al.,

$$p(t) = p_i \left(\frac{p_f}{p_i}\right)^{\frac{t}{t_{max}}}, \tag{8}$$

where p_i and p_f are the initial value and the final value at iteration t_{max}, respectively, for each parameter p. t_{max} in this first series of experiments was set to 500, both because we aimed at having a performant system in that number of iterations, and because of our limited resources.

A simultaneous optimization of the initial and final values for all the parameters was out of our capabilities. Instead, the methodology followed was to first find reasonable values for all of them, by means of trial-and-error search guided by intuition. Next, a systematic optimization was performed, beginning with the simultaneous search for σ_i and σ'_i, followed by σ_f and σ'_f, and ending with c'_i and c'_f. All the process was carried out using a network whose supernets and subnets had $3 \times 3 \times 3$ neurons each. The best values found for the parameters were: $\sigma_i = 10$, $\sigma'_i = 15$, $\sigma_f = 10$, $\sigma'_f = 5$, $c'_i = .1$ and $c'_f = .1$. The same optimal values were found for σ_i and σ_f, and for c'_i and c'_f, because 500 iteration is not a large enough number to take advantage of decreasing position neighborhoods.

The optimized values were afterwards used in a network with six neurons in each dimension. Figure 2a shows the evolution of the gross movement θ' during the learning. Every 4 iterations, learning was interrupted, and the average position and orientation errors over 200 random poses within the workspace were measured using (1). The same random poses were corrected using θ'', and the remaining fine errors are displayed in Figure 2b. The error decreases slowly at this time scale, especially the gross motion errors. However, the fine movements, even from the beginning, are not completely disastrous. This means that the Jacobians of our f^{-1} mapping are not prone to change, even after the robot has suffered a serious and complex damage.

The learning is too slow for our requirements. That was to be expected because this is a difficult task. The previous work with this kind of algorithms attempted, at most, to control three position coordinates and two orientation coordinates. Too see what the effect of adding new dimensions to the problem is, compare the learning curves in [**?**] for the simple positioning of a robot arm with those resulting from controlling also two orientation coordinates. A previous study for this same application [**?**], [**?**] indicated that the original system could not perform much better than 1 cm position error and 1 degree orientation error after 10,000 iterations.

V. Separating neighborhoods

After some analysis of the results obtained with Ritter et al.'s algorithm, it became clear that one of the main problems of the algorithm in this new setting was that the neighborhoods did not reveal to be especially useful. As a matter of fact, the optimal values chosen, $\sigma_i = 10$ and $\sigma'_i = 15$, are low and do not make a great difference with null neighborhoods. Thus, there is no possibility for much cooperation, which is a very important factor for quick learning, as repeatedly noted by the authors of the algorithm.

Our mapping is a 6-variate function that depends on 3 position and 3 orientation coordinates of the end-effector. However, the degree of dependency may not be necessarily the same between all the command components and coordinates. Changes in position coordinates may influence slowly the orientation commands, and the same can be supposed about orientation coordinates with respect to position commands. This advocates for using large gaussians for g_k and g_{kl}. But, on the other hand, position and orientation commands can be guessed to be very sensitive to the real position and orientation coordinates, respectively. This means that, for example, too large neighborhoods in the position coordinates space would be counterproductive to learn position commands.

Figures 3a and 3b seem to support this hypothesis. They represent the final position and orientation error of the system as a function of σ_i and σ'_i, respectively. Figure 3b is the most typical one. It can be seen that the optimum σ'_i value is very different for position and for orientation, implying, for example, that the best value for position has no sense for orientation. It can be seen that the position error behaves smoothly when the optimum is surpassed. On the contrary, for orientation, σ'_i must be restricted to a very short and low range and, if its minimum is overshot, the result is disastrous. In Figure 3a the roles are reversed, position and orientation being very sensitive and insensitive, respectively, to the initial width of the position neighborhood, σ_i. In this case, the minimum coincides for both kinds of errors, but this is a side effect of the particular parameter configuration that has resulted from the optimization process, in which $\sigma_i = \sigma_f$ for 500 iterations. Optimizing the parameters for a higher number of iterations, leads to an optimum for orientation located at higher values.

There is no good solution to these contradictory interests within Ritter et al.'s framework: Both neighborhoods must remain small. To deal with this problem we need to use a finer structure, substituting g_k and g_{kl} by Npo_{kl}, determining the neighborhood in orientation coordinates space that affects position commands; Npp_k, determining the neighborhood in position coordinates space that affects position commands; Nop_k, determining the neighborhood in position coordinates that affects orientation commands; and Noo_{kl}, determining the neighborhood in orientation coordinates that affects orientation commands.

Also the matrix $\mathbf{A}_{kl}$ must be divided following the same scheme into $\mathbf{Ap}_{kl}$, $\mathbf{Ao}_{kl}$, and θ_{kl} into θp_{kl} and θo_{kl}. The learning equations become:

$$\theta p_{ij}^{new} = \theta p_{ij}^{old} + c' \ Npp_k(i) \ Npo_{kl}(j) \ (\theta p^* - \theta p_{kl}), \tag{9}$$

$$\theta o_{ij}^{new} = \theta o_{ij}^{old} + c' \ Nop_k(i) \ Noo_{kl}(j) \ (\theta o^* - \theta o_{kl}), \tag{10}$$

$$\mathbf{Ap}_{ij}^{new} = \mathbf{Ap}_{ij}^{old} + c' \; Npo_{kl}(j) \; Npp_k(i) \; (\mathbf{Ap}^* - \mathbf{Ap}_{kl}), \tag{11}$$

$$\mathbf{Ao}_{ij}^{new} = \mathbf{Ao}_{ij}^{old} + c' \; Noo_{kl}(j) \; Nop_k(i) \; (\mathbf{Ao}^* - \mathbf{Ao}_{kl}). \tag{12}$$

At an intermediate stage in the development of the prototype, these modifications were tested, allowing for wider Npo_{kl} and Nop_{kl} neighborhoods, with a significative increase in learning speed. In this version of the algorithm, the four parameters associated to neighborhoods become eight, and finding good values becomes painful. Fortunately, as explained in Section VIII, subsequent modifications of the algorithm made this decoupling unnecessary. For this reason, we do not present here results with this version of the algorithm.

VI. What to propagate

We introduce now the most fundamental modification to the original algorithm. It will be argued that the way cells cooperate in the original algorithm is not the best one for on-line adaptivity. The modification of (4), (5), (6) and (7) proposed below coincides with the original version applied to the winning neuron, i.e., in the limit of null neighborhoods both versions are minimizing the same quadratic error function. The difference lies only in the information that is propagated to neighboring cells.

A. Coooperation among the θ_{ij}

Ritter et al.'s approach consists in obtaining an estimation θ^* of $f^{-1}(\overline{\mathbf{w}}_{kl})$ based on $\mathbf{A}_{kl}$ and the first movement attempting to attain $\mathbf{u}$. θ^* is immediately assigned to θ_{kl} and, in general, every θ_{ij} is moved towards the same value θ^* in keeping with the closeness of cells ij and kl. We show that this form of cooperation can be improved for changing environments.

Suppose that the original function f^{-1} has been already learned. If a sudden change in the geometry of the robot takes place, we expect the cooperation among units to allow a quick learning of the new f'^{-1} function. Logically, the smallest the change in f^{-1}, the fastest the retraining. But imagine that $f^{-1} = f'^{-1}$. If we increase the neighborhoods, the original function is distroyed.

Consider a modification of the learning rule in which the quantity to be propagated is not θ^*, but the change that θ_{kl} must undergo, that is, $\theta_{ij}^{new} \leftarrow \theta_{ij}^{old} + (\theta^* - \theta_{kl}^{old})$. Thus, (4) and (6) become somewhat simpler:

$$\theta^* = \Delta\theta = \mathbf{A}_{kl}(\mathbf{u} - \mathbf{u}'), \tag{13}$$

$$\theta_{ij}^{new} = \theta_{ij}^{old} + c' \; g_k(i) \; g_{kl}(j) \; \theta^*. \tag{14}$$

Figure 4 illustrates this rule and compares it with the classical one. It is easy to check that this kind of cooperation does not modify the implemented function if $f^{-1} = f'^{-1}$, whatever the width of the neighborhood. Nevertheless, this modification is not better for an arbitrary new function f'^{-1}. The point is that the new version works better when the new function is more similar to the original than to a constant function in the proximity of $\overline{\mathbf{w}}_{kl} = (\mathbf{w}_k, \mathbf{w}_{kl})$. To see this, suppose that we are using σ-wide neighborhoods, such that $g_k(.)g_{kl}(.)$ is approximately 1 in a spherical ball Ω centered on $\overline{\mathbf{w}}_{kl}$. We depart from a network that has already encoded the function f^{-1}, so that every cell ij satisfies $\theta_{ij} = f(\overline{\mathbf{w}}_{ij})$. Now we evaluate the changes made to θ_{ij} by steps (4) and (6) (classical version), and (13) and (14) (new version), when trying to learn the new function f'^{-1}. Let $\theta_{clas}(\mathbf{w})$ and $\theta_{upd}(\mathbf{w})$ be the new values that a hypothetical cell centered on $\mathbf{w}$ would assume as a consequence of the classical and new update versions of the learning rule, respectively. In the ball Ω, $\theta_{clas}(\mathbf{w}) = f'^{-1}(\overline{\mathbf{w}}_{kl})$ and $\theta_{upd}(\mathbf{w}) = f^{-1}(\mathbf{w}) + (f'^{-1}(\overline{\mathbf{w}}_{kl}) - f^{-1}(\overline{\mathbf{w}}_{kl}))$. The goodness of the new and the classical versions can be evaluated by the average error they would cause to cells located in the ball Ω:

$$E_{clas} = \int_\Omega (\theta_{clas}(\mathbf{w}) - f'^{-1}(\mathbf{w}))^2 = \int_\Omega (f'^{-1}(\overline{\mathbf{w}}_{kl}) - f'^{-1}(\mathbf{w}))^2 = \int_\Omega (f'^{-1}(\mathbf{w}) - k_1)^2 \tag{15}$$

$$\begin{aligned} E_{upd} &= \int_\Omega (\theta_{upd}(\mathbf{w}) - f'^{-1}(\mathbf{w}))^2 = \int_\Omega (f^{-1}(\mathbf{w}) + (f'^{-1}(\overline{\mathbf{w}}_{kl}) - f^{-1}(\overline{\mathbf{w}}_{kl})) - f'^{-1}(\mathbf{w}))^2 = \\ &= \int_\Omega (f'^{-1}(\mathbf{w}) - (f^{-1}(\mathbf{w}) + k_2))^2 \end{aligned} \tag{16}$$

where k_1 and k_2 are constants. From this, we can conclude that the new update rule is better when the new function is more similar to the old one (shifted to make them coincident in $\overline{\mathbf{w}}_{kl}$) than to a constant function.

B. Cooperation among the Jacobians

It is not possible to estimate with only two points the ideal Jacobian matrix at $\overline{\mathbf{w}}_{kl}$, but it can be corrected in the direction indicated by the two points. The corrected $\mathbf{A}_{kl}$ matrix is called $\mathbf{A}^*$ and is used as desired matrix by all the $\mathbf{A}_{ij}$. Thus, in all the relevant aspects for us, the problem is the same we encountered with the θ_{ij} update. The corresponding suggested modifications for (5) and (7) are:

$$\mathbf{A}^* = (\Delta\theta - \mathbf{A}_{kl}\Delta\mathbf{u})\frac{\Delta\mathbf{u}^T}{\|\Delta\mathbf{u}\|^2} \tag{17}$$

$$\mathbf{A}_{ij}^{new} = \mathbf{A}_{ij}^{old} + c' \; g_k(i) \; g_{kl}(j) \; \mathbf{A}^*. \tag{18}$$

The discussion is similar to that in the last subsection, and also the conclusions are very similar: The new update version is better when the Jacobian function of f'^{-1} is more similar to $\frac{\partial \vec{f}^{-1}}{\partial \mathbf{w}}$ than to a constant matrix.

The adoption of these modifications leads to a dramatical improvement in the learning efficiency for gross positioning, as can be seen in Figure 5, which of course implies also a speed-up in the learning of fine positioning. As the information propagated from cell to cell is more meaningful, the neighborhoods are allowed to be much wider. The acceleration of the learning shown in our application is especially notable because, although the thetas's are different, the Jacobian matrices are the same for all the cells. As can be deduced from the above explanation and from the direct observation of (5) - (7) and (17) - (18), initially there is no difference between the Jacobian error signals propagated by the two versions, like in learning from scratch. Thus, these results seem to confirm the usefulness of this version in intermediate stages of learning, the system having departed or not from scratch.

VII. Improving the estimation of $f^{-1}(\overline{\mathbf{w}}_{kl})$

Note that the learning of the Jacobians is autonomous, i.e., it does not rely on the accuracy of the θ_{kl}'s. Instead, the learning of the θ_{kl}'s depends on the estimate (4) of $f^{-1}(\overline{\mathbf{w}}_{kl})$, which depends on $\mathbf{A}_{kl}$. If this matrix does not approximate accurately the Jacobian of f^{-1} at $\mathbf{u}'$, θ^* and $f^{-1}(\overline{\mathbf{w}}_{kl})$ can differ substantially, especially if $\mathbf{u}'$ is far from $\overline{\mathbf{w}}_{kl}$. As at the beginning of the learning $\mathbf{A}_{kl}$ is inaccurate, we suggest to take $\theta' = \theta_{kl}$ as first movement, instead of $\theta_{kl} + \mathbf{A}_{kl}(\mathbf{u} - \overline{\mathbf{w}}_{kl})$. Consequently, assuming the modifications described in the last section have been adopted,

$$\theta^* = \mathbf{A}_{kl}(\overline{\mathbf{w}}_{kl} - \mathbf{u}'). \tag{19}$$

This movement will not attain $\overline{\mathbf{w}}_{kl}$, but the hope is that it will bring the arm to a point closer to $\overline{\mathbf{w}}_{kl}$, and thus reduce the dependence on the accuracy of the Jacobians, contributing to a faster learning of the θ_{ij} at the beginning of the training. Of course, this first movement is less precise in attaining $\mathbf{u}$ than $\theta_{kl} + \mathbf{A}_{kl}(\mathbf{u} - \overline{\mathbf{w}}_{kl})$, but we can choose to make it instead of (2) when the purpose of moving the robot to $\mathbf{u}$ is only learning. Thus, from now on, the first and second movements of a learning iteration do not coincide with the gross and fine movements during normal operation. This modification also amelioarates the learning behavior of the Jacobians because of two reasons:

- The original version produces points $\mathbf{u}'$ and $\mathbf{u}''$ which are very near to one another. The short segment $(\mathbf{u}', \mathbf{u}'')$ can be wherever in the receptive field of the cell and, thus, the average derivatives in the receptive field are learned. Instead, with the new version a longer step is made, because $\mathbf{u}'$ is the center of the cell, and $\mathbf{u}''$ can be wherever in the receptive field. As a consequence of the longer steps, less iterations are needed to learn the average derivatives assuming that second derivatives are close to constant within the receptive field of each cell.

- The amount of influence of a cell on another depends on the distance between the cell centers. However, it would be more correct to base it on the distance between the segment $(\mathbf{u}', \mathbf{u}'')$ and the center of the cell. Imagine that, using the classical version, $(\mathbf{u}', \mathbf{u}'')$ is located at the frontier of two receptive fields; the learning should be equal for the two neighboring cells, but it is not. Instead, with the new version, as the segment is more centered, the distance from it and from the winning cell are almost equivalent. Of course, this is relevant only for rather close cells.

Introducing directly this modification only improves the learning slightly. However, it allows for much larger learning rates, because the theta corrections are less noisy. Figure 6 shows the result. The errors displayed are, like in previous figures, averages of random gross movements of the robot during normal operation, $\theta_{kl} + \mathbf{A}_{kl}(\mathbf{u} - \overline{\mathbf{w}}_{kl})$, for random $\mathbf{u}$.

VIII. Schedulings

A. Neighborhood scheduling

When the neighborhood width is large, there is a large number of updated cells per iteration. Few iterations are then required to learn the mapping in all the input space at a coarse level of resolution. Instead, when the neighborhoods are small, the number of cells changing significantly their output in one iteration is very low, and many more iterations are required to make all the cells learn the same number of times as with a larger neighborhood. The neighborhood scheduling proposed by Ritter et al. does not take into account this fact. We will derive now a neighborhood scheduling expressing explicitly all the hypotheses on which it is based. First, we define $L(\overline{\mathbf{w}}_{ij}, \sigma, \sigma')$ as the expected learning for cell i in one iteration using neighborhoods of width σ:

$$L(\overline{\mathbf{w}}_{ij}, \sigma, \sigma') = \int_{\Omega_{pos} \times \Omega_{or}} p(\mathbf{w}) g(\mathbf{w}_{pos}, \mathbf{w}_i, \sigma) g(\mathbf{w}_{or}, \mathbf{w}_{ij}, \sigma') d\mathbf{w}, \tag{20}$$

where Ω_{pos} and Ω_{or} are the position and the orientation spaces respectively of thef f^{-1} domain (the workspace), $p(\mathbf{w})$ is the probability density in Ω_I, and $g(\mathbf{w}_{pos}, \mathbf{w}_i, \sigma)$ is the value in $\mathbf{w}_i$ of a neighborhood centered on $\mathbf{w}_{pos}$ of width σ. Since we want the workspace to be uniformly learned, without 'holes of error', the distribution of the patterns $p(\mathbf{w})$ in the input space is, like in previous works, uniform. Then, for every $\overline{\mathbf{w}}_{ij}$ (except those too close to the border of the input space[2]) the value of L is the same:

$$L(\overline{\mathbf{w}}_{ij}, \sigma, \sigma') = \frac{\sigma^3 \ \sigma'^3 \ (2\pi)^3}{Vol(\Omega_{pos}) \ Vol(\Omega_{or})}, \tag{21}$$

where $Vol(\Omega_{pos})$ $Vol(\Omega_{or})$ are the volumes of position and the orientation space respectively, and a gaussian shape ** has been assumed for g, as usual. $Z(\overline{\mathbf{w}}_{ij}, \sigma, \sigma') = 1/L(\overline{\mathbf{w}}_{ij}, \sigma, \sigma')$ is the mean number of iterations required to have an expected learning quantity of 1 (i.e., the learning underwent by cell ij when just one pattern centered in the cell is learned). Now we must establish how much L is accumulated along time with each σ. Consider this hypothesis: a cell is visited with every neighborhood width a mean number of times proportional to L. That is, the more iterations in average are needed by a cell to learn,

[2]Taking into account these border effects is another possible improvement left for further research.

quicker the neighborhood are decreased. This seems to a certain point illogical and opposed to the rationale stated above. However we can show that this is the assumption underlying the "classical" scheduling. In effect, this hypothesis can be expressed as

$$\frac{\partial t}{\partial Z} = k' L, \tag{22}$$

where k' is a constant. On the other hand,

$$\frac{\partial Z}{\partial t} = \frac{\partial Z}{\partial \sigma}\,\frac{\partial \sigma}{\partial t} + \frac{\partial Z}{\partial \sigma'}\,\frac{\partial \sigma'}{\partial t} = -Z\left(\frac{3}{\sigma)}\,\frac{\partial \sigma}{\partial t} + \frac{3}{\sigma')}\,\frac{\partial \sigma'}{\partial t}\right). \tag{23}$$

Linking (23) and (22), after some manipulations we arrive to the equation system:

$$\frac{\partial \sigma}{\partial t} = -\ k1\ \sigma, \tag{24}$$

$$\frac{\partial \sigma'}{\partial t} = -\ k1\ \sigma', \tag{25}$$

where $k1$ is a constant, whose solution is

$$\sigma(t) = C1\ e^{k1t}, \tag{26}$$

$$\sigma(t) = C2\ e^{k1t}, \tag{27}$$

being $C1 = \sigma(0)$. This kind of scheduling can be easily shown to be equivalent to (8) for a fixed number of iterations. However, to keep the equivalence for different number of iterations, the proportion between the final and the initial value must be recomputed as a function of $tmax$. Another hypothesis, agreeing more with our initially declared intentions, requires a cell to be visited the same mean number of times with every neighborhood width σ. This means that we must stay with each σ a time proportional to $Z(\overline{\mathbf{w}}_{ij}, \sigma, \sigma')$. Thus,

$$\frac{\partial t}{\partial Z} = k' \tag{28}$$

Proceeding like above, from the last hypothesis we get

$$\frac{\partial \sigma}{\partial t} = -\ k''\ \sigma^4\ \sigma'^3, \tag{29}$$

$$\frac{\partial \sigma'}{\partial t} = -\ k''\ \sigma^3\ \sigma'^4, \tag{30}$$

which implies

$$\frac{\partial \sigma}{\partial t} = k2\ \sigma^7, \tag{31}$$

$$\frac{\partial \sigma'}{\partial t} = k2\ \sigma'^7, \tag{32}$$

whose solution is

$$\sigma(t) = (k\ t\ +\ C3)^{-\frac{1}{6}}, \tag{33}$$

$$\sigma'(t) = (k\ t\ +\ C4)^{-\frac{1}{6}}, \tag{34}$$

where the constants $C3$ and $C4$ determine the initial $\sigma(0) = C3^{-\frac{1}{6}}$ and $\sigma'(0) = C4^{-\frac{1}{6}}$. Equations (24-25) and (31-32) can be considered as extreme cases of the family of schedulings $\frac{\partial \sigma}{\partial t} = k2\ \sigma^n$, with $n = 7$ assuming that a cell needs to be visited the same number of times to learn the mapping, whichever the level of coarseness (given by the neighborhood widths), and with $n = 1$ being the old schedulings [[and given relatively much more time to big neighborhoods than to little ones]]. Some experiments showed that $n = 7$ provides more stable learning. Howeve the equitative treatement to all neighborhoods revealed not exactly adapted to reality [or experimental results]. The initial stages of the training require relatively more learning than the normalized number of iterations prescribed by the second assumption (28), because the quality of the learning is poor due to border effects (not taken into account in our derivation) and other collateral effects[3]. We adopted a simple solution, an intermediate scheduling with $n = 4$, which corresponds to the hypothesis $\frac{\partial t}{\partial Z} = k' L^{\frac{1}{2}}$, giving raise to this solution:

$$\sigma(t) = (k\ t\ +\ C5)^{-\frac{1}{3}}, \tag{35}$$

$$\sigma'(t) = (k\ t\ +\ C6)^{-\frac{1}{3}}, \tag{36}$$

[3]At an intermediate stage of the learning, the function can be supposed well approximated in the inmediate superior level of coarseness. Instead, when learning begin this is not true. Besides, the estimation of the function made by the superposition of two touching neighborhoods is better when their scale is small

To homogenize the comparisons we have manipulated the initial conditions indirectly by taking $C5 = \sigma_i^{-3}$ $C6 = \sigma_i'^{-3}$. Thus, two parameters, σ_i and k (which replaces σ_f), determine a neighborhood scheduling.

It must be stressed that k is dependent on $Vol(\Omega_I)$. Thus, taking this into account, the optimized k for a region can be updated without further parameter search to a more restricted or enlarged region, or to a different one with similar characteristics.

B. *Learning rate scheduling*

It is known that, to ensure convergence to a global optimum, stochastic minimization requires to decrease linearly the learning rate [**?**]. Nevertheless, other studies suggest that, decreasing more slowly the learning rate at the beginning of the minimization, the search is quicker. A review of learning rate scheduling for stochastic search can be found in [**?**]. Whatever the scheduling chosen, it must be noted that an iteration in these schedulings does not correspond to an iteration in each of the cells in our system. Instead, to accomplish the learning corresponding to one iteration in a single-cell setting, each of our units has to wait $Z(\mathbf{w}_i, \sigma)$ iterations. Denoting by T and t the time measured in a single-cell setting, and in our system, respectively,

$$\frac{\partial c'}{\partial t} = \frac{\partial c'}{\partial T}\,\frac{\partial T}{\partial t} = \frac{\partial c'}{\partial T}\; L(\sigma), \tag{37}$$

where $\partial c'/\partial T$ is the derivative of the standard learning rate schedulings (for a single cell).One could theoretically optimize the parameter of a standard learning rate scheduling for a cell of the network, and apply (37) directly to the complete system.

Figures 7a and 7b show the result of applying the new parameter schedulings. The final error improvement is not very important with respect to the preceding figure because of two reasons:

-In all the experiments presented in this paper $c'_i = c'_f$ because, in the short time scales we are exploring, there was no significative improvement in decreasing c' with time. Thus, with a constant scheduling, there is no difference for c' between the two versions.

- Our neighborhood scheduling does not have a very different shape from the scheduling used for all parameters in previous works, $p(t) = p_i\ (\frac{p_f}{p_i})^{\frac{t}{t_{max}}}$.

However, there is more than efficiency to be gained with these schedulings. First, the learning is more stable in Figure 7. On the other hand, all the parameters in the experiment of the preceding section were tuned to minimize the error for 500 iterations. But during the first half of the learning, the error in that figure is double than that in our last version. In fact a certain time-independence in the parameter optimization is obtained with the new schedulings. One of the burdens of the original system was that the tuning of the parameters at different time scales produced very different results and, thus, keeping during a large number of iterations the parameters which where optima at a short time scale often leads to very poor results.

Another very interesting side effect, whose responsibility is shared with the modifications in Section VI, is the virtual elimination of all neighborhood parameters, except those needed for a unified one. In Section V we warned that, to have a really performant system, four different neighborhoods were required, each with two parameters. For example, the values of σ' that minimized the orientation and position errors were different, and overall, half of the optima were narrow and very sensitive to shiftening towards higher values. Instead, in our final system, both kinds of error have very flat areas of quasi-optima, and these areas are very large compared to the absolute value of the optimal neighborhood. Compare the scale of the vertical axes in Figures 8a and 8b with that in Figures 3a and 3b. This shape implies not only that the optimization for each of the parameters is very easy, but also that the four neighborhoods needed in the original system have a large common flat area of quasi-optima values. Thus, only one neighborhood is enough for all purposes without losing precision. The dependency of the position error on the unified neighborhood parameters σ_i and k is shown in Figure 9. It is clear not only that the optima for σ_i and k are very robust, but also that the optimization of k is almost independent of σ. For our neighborhood scheduling to be effective, a wide neighborhood must be beneficial at the beginning of the learning. For this reason, the modification suggested in Section VI or, at least, the separation of neighborhoods indicated in Section V, have to be applied previously. It is also convenient to apply before the modification in Section VII, so that the theta corrections are scarcely noisy almost from the beginning, and the neighborhood distance for Jacobians is truly meaningful (because of the centering).

IX. Tests with the real robot

The recalibration system has been installed in the REIS robot included in the space-station mock-up located at D.A.S.A.'s R&D laboratory, in Bremen, Germany. Figure 10 is a photograph of such a set-up, where the different racks containing the electronic cards that the robot should maintain are shown. The robot must reach the handles of the racks with enough precision to be able to pull them out and, afterwards, extract a faulty card in order to replace it by another one. Notice that there is no problem in adding specific suppletory tools in those positions of the workspace that present special difficulties. This is the case of the right handle in the lower rack, which is just at the border of the region reachable by the manipulator, and a stick has been attached to the handle in order to ease the task of the robot.

Two workspaces are planned to be used in the space station and are being tested in the mock-up: one in front of the racks, as we have just specified, and another one on top of the horizontal surface located just in front of the robot, where it has to be able to manipulate tools as well as rock samples collected by other robots. Both workspaces are simplified instances of the one used in our simulations (described in Section IV.B) in that a fixed orientation of the end-effector is maintained throughout them (perpendicular to the horizontal plane, in the second case, and forming a 60-degree angle with that plane, in the first case). Of course, the system has to compensate for both position and orientation errors. The dimensions of the first workspace are $20 \times 40 \times 40$ cm, and those of the second are exactly the same as those used in the previous simulations.

Testing with the real robot was unavoidably limited by the fact that the integrity of the robot had to be preserved. We were not allowed to introduce local deformations in the robot geometry (link bending, encoder shifting, etc.) by obvious reasons. Thus, so far only the performance of the recalibration system in front of transformations of the robot as a whole (i.e., changes in the relative position of the robot with respect to the environment and, in particular, the cameras) have been tested. These are rigid transformations plus scaling. Note that, because of the independence of rotations with respect to translations, the Jacobians for orientation remain correct after this type of transformations and, therefore, the orientation errors get corrected in just one time step.

For this reason, we are presenting here only position correction results.

Figure 11 shows the evolution of the error for one of the tests performed. The curves were very similar in all the experiments we carried out and, thus, this can be seen as a representative one. This particular curve corresponds to a "global deformation" consisting of a translation of 1, 2 and 3 cm along the x, y and z axes, and a rotation of 2, 3 and 4 degrees around these axes centered at the robot base. Moreover, the position coordinates supplied by the vision module were scaled by a factor of 0.9. This leads to an initial average positioning error of more than 9 cm for gross motion in operation mode.

Note that this figure plots a measure different from those corresponding to simulated experiments. Here, the error for each single second learning movement is displayed (remember that the first does not attempt to reach $\mathbf{u}$), while in simulation averages over 200 movements every 25 learning steps were shown. The reason is clear: the motion of the robot is much slower than the simulated one, and we could not afford making 200 movements every 25 learning steps just to record the evolution of the error for each single test. Instead, after the 500 iterations, we recorded the average position error for 200 commanded robot poses. In the particular test of the figure, this error was .6 mm, it being also representative of the average error in the other tests performed.

X. Conclusions

We have presented a neural network system to recalibrate robots, inspired in Ritter et al.'s work, which can be applied in more practical settings than the original algorithm because of two reasons. First, the system can work without substituting the original controller in commercial robots. And second, the learning is much faster due to the improved cooperation among the learning units.

To attain the first objective we made use of an external module which provided us with the real pose coordinates, in such a way that the mapping from pose coordinates to controller commands could be learned. With this approach we do not have to worry about the ill-conditioning of the Jacobian in possible singularity points, because we are not learning the inverse kinematics, but a bijective function. Besides, the mapping for the non-damaged robot is known to be the identity one and need not be learned. It is possible to modify slightly this approach, learning for instance the mapping from camera coordinates to controller commands, still without replacing the original controller, but then the other advantages are lost.

Ritter et al.'s algorithm was too slow for our requirements, due mainly to the limited neighborhood widths allowed by the system. The most natural and immediate solution is to split the two original neighborhoods into four different ones, duplicating, however, the number of parameters. The improvements made to the algorithm, affecting the cooperation among units, the selection of training points, and the scheduling of parameters, provide the following advantages:

- Much faster learning, more adequate for on-line adaptivity.

- The splitting of the two neighborhoods required in the original system is avoided. Even more, the neighborhoods can be reduced to one, with the consequent elimination of parameters.

- The two remaining neighborhood parameters are more robust and less interdependent. Their selection is more independent of the time scale.

In the present version of the algorithm, the placement of the units is fixed from the beginning, because the inputs are cartesian coordinates, not subject to change. Thus, the algorithm amounts to the learning of the scalar and Jacobian values of a regular grid or table. The exact learning of this table requires 7 movements per cell in the network (one for the θ_{ij} and six to find the Jacobian). In high-dimensional spaces, even with few cells in each dimension, the effort is excessive. For example, in our case, with six units in each dimension, 362,592 movements would be needed (5,103 in the case of only 3 units in each dimension) When the kinematics of the robot change, even slightly, the complete series of movements should be repeated to obtain the exact mapping. Our system has shown to require very few movements to get good approximations: a remaining average error of 1 mm and 0.1 degrees after 25 movements, and of about .3 mm and .03 degrees after 100 movements, for fine motion.

We think that other local learning algorithms of the SOM family as well as neural network algorithms based on local units, such as [?] or [?], can also benefit from the innovations introduced in this paper.

The recalibration system has been installed in the space-station mock-up installed in

D.A.S.A.'s R&D laboratory, in Bremen, Germany. Results of the installation tests have been presented. We do not have records of long-term operation in the mock-up yet, but we hope to have them in the near future. Obtaining records of operation in space depends on the future plans of D.A.S.A.

There is still room for improvement in our system. One of the main ways is dealing specifically with border effects in neighborhoods. This will ultimately lead to individual learning rates and neighborhood schedulings for each cell. Moreover, we have begun to test a more sophisticated selection of sample movements with very promising results. Although the workspaces are fixed, it can be interesting to move cells according to the difficulty of the mapping instead of minimizing the quantization error. The averaging of output cells used in [?] can be readily applied, although its realization is more complex and cumbersome in a hierarchical system. Finally, it could be interesting to test other hypotheses for the accumulation of L along the σ range, especially in the case that the cells move or other sources of error are present in the initial learning stages, like the selection of $\mathbf{u}'$ made in the original algorithm.

Acknowledgments

This work was partially supported by the ESPRIT III Program of the European Union under the contract N. 6715 (project CONNY). We thank Albrecht Kellner, Eike Schmidt and Bernd Maediger, from Daimler-Benz Aerospace, for providing the application specifications, the robot set-up, as well as fruitful discussions all along the development of the implementation. We also like to thank Enric Celaya for hepful discussions, Gabriela Cembrano for support and encouragement, Conor Doherty for providing an initial version of the extended Kohonen-maps program, and Bernd Maediger for supplying the photograph in Figure 10.

Figure captions

Figure 1. A previous approach to recalibrate robot arms is the direct learning of the Inverse Kinematics mapping. However, if a controller already exists, this learning is not necessary for a non-damaged robot, because Controller($\mathcal{X}$) and Inverse Kinematics(X) coincide, an thus Forward Kinematics(Controller($\mathcal{X}$))= $\mathcal{X}$.

Figure 2a. Evolution of the average gross positioning errors along time with Ritter et al.'s algorithm. Gross positioning errors are those produced using $\theta' = \theta_{kl} + \mathbf{A}_{kl}((\mathbf{u_p}, \mathbf{u_o}) - (\mathbf{w}_k, \mathbf{w}_{kl}))$. The average is over 200 $\mathbf{u}$ samples within the workspace.

Figure 2b. Evolution of the average fine positioning errors along time with Ritter et al.'s algorithm. Fine positioning errors are those produced using $\theta'' = \theta'_{kl} + \mathbf{A}_{kl}(\mathbf{u} - \mathbf{u}')$. θ' and $\mathbf{u}$ are those used in Figure 2a.

Figure 3a. Sensitivity to the initial neighborhood width in Ritter et al.'s algorithm, displayed as the final error reached as a function of σ_i. σ_f is also manipulated to hold the proportion between σ_i and σ_f, found in the optimization of the parameters, constant.

Figure 3b.Sensitivity to the initial neighborhood width in ritter et al.'s algorithm, displayed as the final error reached as a function of σ'_i.

Figure 4. Changes in the θ values produced by the original Ritter et al.'s algorithm and by our new cooperation rule. To simplify the figure, the value of the neighborhood functions in the surroundings of the winning cell has been assumed to be one.

Figure 5. Evolution of average gross motion errors with the cooperation rule suggested in Section VI.

Figure 6. Evolution of average gross motion errors with the modifications suggested in Sections VI and VII.

Figure 7a. Evolution of average gross motion errors in our final system, with all the modifications adopted.

Figure 7b. Evolution of fine motion errors in our final system, corresponding to a visual-based feedback correction to the gross movements in Figure 7a.

Figure 8a. Sensitivity to the initial neighborhood width in our final system, displayed as average gross error as a function of σ_i.

Figure 8b. Sensitivity to the initial neighborhood width in our final system, displayed as average gross error as a function of σ'_i.

Figure 9. Sensitivity of our final system: Average gross motion errors reached by our final system as a function of the k and σ_i parameters of the unified neighborhood resulting from the unification of those for position and orientation.

Figure 10. The REIS robot and its set-up at D.A.S.A.'s R&D laboratory.

Figure 11. On-line errors produced after the introduction of a global deformation in the visual data. Although it is not well appreciated in the graphic, the first on-line error is approximatedly 2 cm. See text for details.