Automatic recalibration of a space robot: an industrial prototype*

Vicente Ruiz de Angulo and Carme Torras

Institut de Cibernètica (CSIC-UPC). Diagonal 647, 08028-Barcelona. Spain
e-mail: ruiz@ic.upc.es, torras@ic.upc.es

Abstract. We present a neural network method to calibrate automatically a commercial robot after undergoing wear or damage, which works on top of the nominal inverse kinematics embedded in its controller.

1 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, Daimler-Benz Aerospace proposed an application of maintenance of electronic equipment in a space station that required the automatic recalibration of a 6-dof robot in-situ after wear had occurred. We present here the solution that was implemented in the final demonstrator for the project.

Our starting point was the work of Ritter et al. [2, 1, 5] on learning inverse kinematics from scratch using a hierarchical self-organizing map (SOM). We have modified this model in several ways to suit a more practical setting. Instead of learning the whole mapping, our algorithm learns only the appropriate corrections with respect to the inverse kinematics embedded in the controller, which is thus maintained. The other modifications enhance the cooperation between neurons, speeding up learning by a factor of 70 (near to 2 orders of magnitude).

A detailed description of the methods and results succinctly presented in this paper can be found in [3].

2 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 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,

* This work was partially supported by the ESPRIT III Program of the European Union under the contract No. 6715 (project CONNY). We thank Enric Celaya for hepful discussions, Gabriela Cembrano for support and encouragement, and Conor Doherty for providing an initial version of the extended Kohonen-maps program.

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 the weights $\mathbf{w}_k$ and $\mathbf{w}_{kl}$.
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:

$$\theta^* = \theta_{kl} + \mathbf{A}_{kl}(\mathbf{u} - \mathbf{u}'). \tag{2}$$

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

$$\mathbf{A}^* = \mathbf{A}_{kl} + (\Delta\theta - \mathbf{A}_{kl}\Delta\mathbf{u})\frac{\Delta\mathbf{u}^T}{||\Delta\mathbf{u}||^2}. \tag{3}$$

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

$$\theta_{ij}^{new} = \theta_{ij}^{old} + c'\ g_k(i)\ g_{kl}(j)\ (\theta^* - \theta_{ij}), \tag{4}$$

and the Jacobian matrix:

$$\mathbf{A}_{ij}^{new} = \mathbf{A}_{ij}^{old} + c'\ g_k(i)\ g_{kl}(j)\ (\mathbf{A}^* - \mathbf{A}_{ij}), \tag{5}$$

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

3 A new application: Inverse kinematics update

Our application entails learning a mapping from desired robot poses to appropriate pose commands which, when supplied to the controller, lead to the attainment of those desired poses. For the intact robot, this mapping is the identity. After some degradation, this mapping amounts to sending the robot to a fake pose in order for it to reach the desired one. Thus, $\mathbf{u}$ and θ represent for us 6D vectors denoting pose coordinates and pose commands, respectively. This approach avoids the problem of the original application of having a multivalued inverse function, because the controller always chooses the same joint angles for a fixed command. Although we are still learning an inverse function, we now know the workspace shape, so that we can directly place the centers of the cells in a regular grid covering it, in order to minimize the quantization error. These centers do not need to move if the workspace shape does not change and, thus, the first step of applying the Kohonen rule is eliminated from the algorithm.

The direct application of Ritter et al.'s algorithm, with this slight modification, to our particular setting was not as quick as expected, even if the neighborhood widths were tuned in order to maximize learning speed. There are several reasons for this, which will be explained in the following sections together with the modifications triggered by each of them.

4 Separating dependencies

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 is not the same between all the command components and coordinates. Due to the characteristics of the workspace used (far from singularities and of small size relative to that of the robot) changes in position coordinates influence very slowly the orientation commands, and the same can be said 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 are very sensitive to the real position and orientation coordinates, respectively. This means that, for example, too large neighborhoods in the position coordinates space are counterproductive to learn position commands.

Since there is no good solution to these contradictory interests within Ritter et al.'s framework, we decided to use two different hierarchical networks: one with a narrow g_k and a wide g_{kl} to compute only position commands, and another with a wide g_k and a narrow g_{kl} to compute only orientation commands. Moreover, the position network needs less units in the supernet, while the orientation network saves neurons at the subnet level. This modification leads to a significant increase in learning speed.

As all the subsequent modifications to the algorithm do not distinguish between position and orientation commands, in the remaining of the paper we will follow the notation in Section 2 to refer indistinctly to both networks.

5 What to propagate

Now we show that, in the usual case in which the kinematics change is not drastic, the information propagated to neighboring units can be modified to improve their cooperation.

Coooperation among the θ_{kl}. Ritter et al.'s approach consists in obtaining an estimation θ^* 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_{rs} is moved towards the same value θ^* in keeping with the closeness of cells rs and kl. 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, $\theta_{rs}^{new} \leftarrow \theta_{rs}^{old} + (\theta^* - \theta_{kl}^{old})$. Thus, (2) and (4) become somewhat simpler:

$$\theta^* = \Delta\theta = \mathbf{A}_{kl}(\mathbf{u} - \mathbf{u}'), \tag{6}$$

$$\theta_{ij}^{new} = \theta_{ij}^{old} + c'\ g_k(i)\ g_{kl}(j)\ \theta^*. \tag{7}$$

It is easy to prove that this kind of cooperation works better than Ritter et al.'s when the new function is more similar to the original one 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^2-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, so that every cell rs satisfies $\theta_{rs} = f(\overline{\mathbf{w}}_{rs})$. Now we evaluate the changes made to θ_{rs} by steps (2) and (4) (classical version), and (6) and (7) (new version), when trying to learn the new function f'. 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. 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:

$$E_{clas} = \int_\Omega (\theta_{clas}(\mathbf{w}) - f'(\mathbf{w}))^2 = \int_\Omega (f'(\mathbf{w}_k) - f'(\mathbf{w}))^2 = \int_\Omega (f'(\mathbf{w}) - k_1)^2$$

$$E_{upd} = \int_\Omega (\theta_{upd}(\mathbf{w}) - f'(\mathbf{w}))^2 = \int_\Omega (f(\mathbf{w}) + (f'(\mathbf{w}_k) - f(\mathbf{w}_k)) - f'(\mathbf{w}))^2 = \int_\Omega (f'(\mathbf{w}) - (f(\mathbf{w}) + k_2))^2,$$

where k_1 and k_2 are constants.

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 $\mathbf{A}_{rs}$. Thus, in all the relevant aspects for us, the problem is the same we encountered with the θ_{rs} update. The corresponding suggested modifications for (3) and (5) are:

$$\mathbf{A}^* = (\Delta\theta - \mathbf{A}_{kl}\Delta\mathbf{u})\frac{\Delta\mathbf{u}^T}{\|\Delta\mathbf{u}\|^2} \tag{8}$$

$$\mathbf{A}_{ij}^{new} = \mathbf{A}_{ij}^{old} + c'\ g_k(i)\ g_{kl}(j)\ \mathbf{A}^*. \tag{9}$$

The discussion is similar to that in the last subsection: The new update version is better than the classical one when the Jacobian function of f' is more similar to $\frac{\partial f}{\partial \mathbf{w}}$ than to a constant matrix.

6 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 $L(\mathbf{w}_r, \sigma)$ as the expected learning for cell r in one iteration using neighborhoods of width σ:

$$L(\mathbf{w}_r, \sigma) = \int_{\Omega_I} p(\mathbf{w}) h(\mathbf{w}, \sigma, \mathbf{w}_r) d\mathbf{w} = \frac{\sigma^n \ (2\pi)^{n/2}}{\text{Volume}(\Omega_I)}, \tag{10}$$

where Ω_I is the n-dimensional input space of f, $p(\mathbf{w})$ is the probability density in Ω_I, and $h(\mathbf{w}, \sigma, \mathbf{w}_r)$ is the value in $\mathbf{w}_r$ of a neighborhood centered on $\mathbf{w}$ of width σ. The second equality results from assuming Gaussian neighborhoods and a uniform $p(\mathbf{w})$, and it is valid for $\mathbf{w}_r$'s not too close to the border of Ω_I.

Now we must establish how much L should be accumulated along time with each σ. We assume the simplest hypothesis, that is, a cell is visited the same mean number of times with every neighborhood width σ. This means that we must stay with each σ a time inversely proportional to $L(\mathbf{w}_r, \sigma)$.

7 Experimental results

Our recalibration system was tested on a Reiss robot in the space-station mock-up installed in Daimler-Benz Aerospace, Bremen. 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. Evaluation with the real robot was necessarily much more restricted than that carried out in simulation. Since the robot integrity was to be preserved, only decalibrations consisting of translations and rotations of the whole robot were tested. The results were in complete agreement with the simulation results for the same situation [3].

Due to lack of space, we will present here only simulation results obtained for a more interesting and representative case: one in which the geometry of the robot undergoes serious distortion. 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.

Figure 1 shows the results of performing 200 iterations with our learning system, where supernets and subnets had $3 \times 3 \times 3$ neurons each. Every 4 iterations, learning was interrupted, and the average position and orientation errors over 200 random poses of the workspace were measured using (1). We must remark that these results were obtained with two hierarchical networks which, as a whole, had the same number of neurons as the Ritter et al.'s model we used for comparison. The parameters were selected after a very rough search, using the same initial σ for all neighborhoods and applying a slow linear decrease to α. The original Ritter et al.'s algorithm with optimized parameters required more than 10,000 iterations to get the same combination of orientation and position errors, as reported in [4].

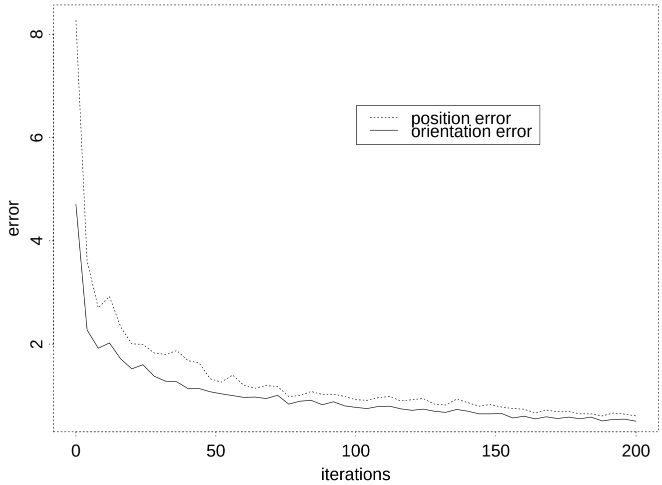

Fig. 1. Evolution of the error along the first 200 iterations. Position and orientation errors are measured in centimeters and degrees, respectively.

8 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. We think that other neural network algorithms based on local units can also benefit from these improvements.

References

1. Martinetz T. and Schulten K.J., 'Hierarchical neural net for learning control of a robot's arm and gripper', *Proc. Intl. Joint Conf. on Neural Networks (IJCNN'90)*, Vol. II, 747–752, San Diego, 1990.
2. Ritter H., Martinetz T. and Schulten K.J., *Neural Computation and Self-Organizing Maps*, New York: Addison Wesley, 1992.
3. Ruiz de Angulo V. and Torras C., 'Automatic recalibration of a space robot: an industrial prototype', *Technical Report IC-DT-03.96*, Institut de Cibernètica (CSIC-UPC).
4. Torras C., Cembrano G., Millán J. del R. and Wells G., 'Neural approaches to robot control: Four representative applications', *Proc. 3rd Intl. Workshop on Artificial Neural Networks (IWANN'95)*, Málaga, June, 1016–1035 (1995).
5. Walter J.A. and Schulten K.J., 'Implementation of self-organizing neural networks for visuo-motor control of an industrial arm', *IEEE Trans. on Neural Networks*, Vol. 4, No. 1, January 1993.