Reducing 6D inverse kinematics to the learning of 3D functions

Vicente Ruiz de Angulo and Carme Torras*

**Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Llorens i Artigas 4-6, 08028-Barcelona, Spain. e-mail:{ruiz, torras}@iri.upc.es*

Abstract. The Inverse Kinematics (IK) of a robot arm is a very complex mapping, with a relatively high dimensionality, which makes its accurate approximation with a learning system very difficult. This is reflected in the high number of training samples, i.e., robot movements, that must be generated, it being unaffordable in many applications. We propose a trick, valid for most commercial robots that greatly reduces the number of movements needed to learn or relearn the IK to a given precision level. This trick can be used with any learning algorithm and consists in expressing the IK as a composition of learnable functions, each having half the dimensionality of the original mapping. Some preliminary results suggest that the time savings thus obtained scale polynomically with the precision required.

1 Introduction

Learning the Inverse Kinematics Mapping (IKM) is an alternative to its calculation from the manufacturer's parameter specification of the robot. It is useful for robots with uncertainties difficult to model, as, for example, robots with flexible links. It is also interesting when the IKM is difficult or slow to compute, as in the case of redundant robots. In general, every robot suffers from wear-and-tear and can even be damaged, therefore needing recalibration from time to time. In particular, high-precision robots may need to be recalibrated often, which makes some applications impractical or impossible. Furthermore, in space stations or dangerous zones, a human could not be available when a recalibration is needed. Thus, on-line learning of the IKM may be very helpful in these cases.

However, it has some drawbacks, as can be the requirement of a sophisticated set-up, with sensors able to determine the position and/or orientation of the end-effector. This set-up could make the system more expensive, but could also be useful for other purposes. A more serious drawback is the number of points required to approximate the mapping up to the desired accuracy.

Typically, neural network applications have many input variables, some of which are redundant, and others have a negligible effect on the output variables. Thus, the underlying mapping can be considered as lying on a low dimensional manifold. The hard part of the learning task is to guess the structure of the mapping from the tangle of information. The difficulty lies here, rather than in the fine approximation of every detail, since the mappings are often fairly simple.

Instead, in the learning of the IKM, one has completely independent input variables, each of them influencing powerfully the result. Under these conditions, the number of points required to approximate the mapping tends to be exponential in the number of variables. Moreover, in contrast with other applications, the mapping has a complex shape and should be approximated with a high accuracy. Thus, the number of learning points required may be huge (Martinetz et al., 1990; Kröse and van der Smagt, 1993).

Several attempts have been made at reducing the number of required samples, among them the use of hierarchical networks (Ritter et al., 1992; Walter and Schulten, 1993), the learning of only the deviations from the nominal kinematics (Ruiz de Angulo and Torras, 1997), and the use of a continuous rep-

resentation by associating a basis function to each neuron (Walter and Ritter, 1996).

In this paper, we propose a practical trick that can be used in combination with all the methods above. It consists in decomposing the learning of the IKM into several independent tasks. This is done at the expense of sacrificing generality: the method works only for some robot models subject to certain types of deformations. Specifically, the method assumes that the last three robot joints cross at a point. This is fulfilled by the more popular commercial robot arms, such as the PUMA or the Stanford robot. The condition continues to hold after any encoder miscalibration and the other most likely deformations of the robot geometry.

The gain obtained is worth the sacrifice. The dimensionality of each of the tasks resulting from the decomposition is half that of the original one. Thus, for a given desired accuracy, Acc, the number of learning points required to learn the IKM directly is $O(Acc^{Dim})$, while through the decomposition it reduces to $O(Acc^{Dim/2})$. This yields an enormous reduction in the number of points required for high-precision applications.

The paper is structured as follows. In the next section we describe the proposed decomposition of the IKM. Section 3 presents algorithms to learn the functions resulting from such decomposition, under different approaches ranging from independent learning to on-line learning. Some preliminary results are presented in Section 4. Finally, some conclusions are drawn in Section 5.

2 Decomposing the Inverse Kinematics Mapping

As mentioned in the preceding section, our purpose is to decompose the learning of inverse kinematics in such a way that we have to approximate functions with an input dimensionality half that of the original function. We will formulate our method for the most common case, a robot arm with 6 rotational degrees of freedom, although it can be applied also to other types of robots. Our work relies on the assumption that the axes of the last three joints cross at a point, which holds for most commercial robot arms.

Let $\theta = (\theta_1, \theta_2, \theta_3)$ and $\nu = (\nu_1, \nu_2, \nu_3)$ be the value of the first three and the last three joint angles, respectively. We begin by explaining why θ can be easily obtained under the above assumption, and then we show how ν can be calculated as a composition of functions dependent on θ.

2.1 Calculus of θ

Let X and Ω be the position and orientation of the end-effector. Our purpose is to express θ as a composition of functions dependent on part of the given data (X, Ω), so that the component functions needing to be learned have an input dimensionality less than or equal to three.

The position X^* of the point at which the last three axes cross can be recovered from X and Ω as follows:

$$X^* = X - \Delta X(\Omega), \tag{1}$$

where $\Delta X(.)$ is a well-defined function, which for each end-effector orientation provides the relative position of X with respect to X^*.

Note that X^* is not moved by varying ν, and thus it depends only on θ. We now like to obtain the inverse function, namely θ as a function of X^*. This is straightforward, except in the cases where the forward kinematics mapping is not one-to-one.

For example, the PUMA robot has 8 configurations resulting from the combinations of right/left arm, up/down forearm, and flip/noflip position of the third link. Multivalued inverse functions can be avoided by considering only one solution branch (e.g., one of the 8 configurations of the PUMA robot).

When the number of joints in θ is greater than three, the situation is affordable with a learning architecture able to associate X^* with a subspace of θ, but one should restrict the workspace to (X, Ω) such that for every solution θ available for X^*, the compatible Ω are the same.

Either way we get independence of θ with respect to Ω, and we can write

$$\theta = \tau(X^*).$$

Using (1), we obtain:

$$\theta = \tau(X - \Delta X(\Omega)), \tag{2}$$

where both $\tau(.)$ and $\Delta X(.)$ are 3D functions.

2.2 Calculus of ν

To simplify the exposition in this section, we will consider by now that Ω is a rotation matrix.

First we define a fixed configuration of the first three joints, θ_0, to be used as reference. Then, we define a new function $\Omega_0(\cdot)$ such that $\Omega_0(\theta)$ is the rotation that transforms the orientation of the end-effector at a configuration (θ, ν) to the orientation it would have at (θ_0, ν):

$$\Omega_0(\theta)\ \Omega(\theta, \nu) = \Omega(\theta_0, \nu). \tag{3}$$

Note that $\Omega_0(.)$ is independent of ν and the only requirements are that the range for ν at every θ is contained in that at θ_0, and that the last links and joints in ν are not flexible.

end-effector [ultimos

We shall now define the function $\phi_0(.)$ such that $\phi_0(\Omega)$ is the ν value which at θ_0 yields the orientation Ω.

in this case

We can apply $\phi_0(\cdot)$ to both members of the equality (3), leading to:

$$\phi_0(\Omega_0(\theta)\ \Omega(\theta,\nu)) = \phi_0(\Omega(\theta_0,\nu)),$$

and thus,

$$\phi_0(\Omega_0(\theta)\ \Omega) = \nu. \tag{4}$$

2.3 The target decomposition

Supposing we are able to learn $\tau(.)$, $\Delta X(.)$, $\phi_0(.)$ and $\Omega_0(.)$, the inverse kinematics can be calculated in two phases. First we obtain θ following equation 2, and then we calculate ν according to equation 4.

We have obtained expressions for θ and ν as a composition of functions, each having as domain a part of the input, X or Ω. However, ϕ_0 and Ω_0 have respectively as input and output a rotation matrix, which could be compacted as Eulerian angles. Thus, by redefining adequately ϕ_0 and Ω_0, we can transform (4) into:

$$\phi_0(Eul[\Omega_0(\theta)\ \Omega]) = \nu, \tag{5}$$

$Eul[R]$ being the eulerian expression of rotation matrix R.

Now, not only $\tau(.)$ and $\Delta X(.)$, but also $\phi_0(.)$ and $\Omega_0(.)$ are 3D functions. Thus, their learning can be expected to require a number of samples orders of magnitude lower than that needed to learn the whole IKM directly.

3 Learning

The function $\Delta X(.)$ is a special case because of its simplicity, and will be considered separately from the other three functions. If, through external sensors, the set-up permits acquiring the position X^* at which the last three axes cross, then this function is not even needed: it suffices to consider (X^*, Ω) directly as input. If, on the contrary, X^* needs to be derived from (X, Ω), then a simple procedure entailing only the motion of the last two joints can be applied. One can observe the position and orientation of the end-effector and then make a step, e.g. in ν_3, whilst maintaining the remaining joints fixed, and again observe the position and orientation. From these two observations the axis of ν_3 can be deduced uniquely. Making a step in ν_2 and another observation, the axis of ν_2 can be deduced. Finally, as neither the observations can be expected to be accurate, nor the actual axes may really cross at a point, the middle point of the segment realizing the minimum distance between the two axes is taken as an estimation of the crossing point. Thus, the total number of observations required is three (which can be integrated in the learning of ϕ_0, as described below), and neither in this case, nor in the previous one, iterative learning is required for the computation of ΔX.

The remaining functions $\tau(.)$, $\Omega_0(.)$ and $\phi_0(.)$ are inverse functions, in the sense that we cannot generate the output for a given input. Their learning can be accomplished with strategies entailing different degrees of parallelism and sophistication, as shown next.

3.1 Independent learning

The simplest approach is to learn each function independently in a phase preceding the functional operation of the robot. Algorithms to provide inputs and outputs for $\tau(.)$, $\Omega_0(.)$ and $\phi_0(.)$ to a learning system are sketched below.

Learning of τ
Repeat for $i = 1$ to whatever
Select $\theta^{i)}$
Choose $\nu^{i)}$ arbitrarily
Move to $(\theta^{i)}, \nu^{i)})$. Observe $X^{i)}$, $\Omega^{i)}$
Learn with $X^{i)} - \Delta X(\Omega^{i)})$ as input and $\theta^{i)}$ as output

Learning of Ω_0
Select ν' arbitrarily
Move to (θ_0, ν'). Observe $\Omega^{0)}$
Repeat for $i = 1$ to whatever
Select $\theta^{i)}$
Move to $(\theta^{i)}, \nu')$. Observe $\Omega^{i)}$
Learn with $\theta^{i)}$ as input and $Eul[\Omega^{0)}(\Omega^{i)})^T]$ as output

Learning of ϕ_0
Repeat for $i = 1$ to whatever
Select $\nu^{i)}$
Move to $(\theta_0, \nu^{i)})$. Observe $\Omega^{i)}$
Learn with $Eul[\Omega^{i)}]$ as input and $\nu^{i)}$ as output

3.2 Partially overlapped learning

There are more efficient alternatives to the independent learning of all functions. We first suggest the parallelization of the learning of ΔX (if X^* cannot be directly observed) and ϕ_0 in one phase, and that of τ^* and Ω_0 in another phase.

For the first parallelization, it is enough that two of the movements carried out in the course of ϕ_0 learning change consecutively ν_2 and ν_3 alone.

The second phase is carried out with the following algorithm:

Learning of Ω_0 and τ
Select ν' arbitrarily
Move to (θ_0, ν'). Observe $\Omega^{0)}$
Repeat for $i = 1$ to whatever
Select $\theta^{i)}$
Move to $(\theta^{i)}, \nu')$. Observe $(X^{i)}, \Omega^{i)})$
Learn Ω_0 with $\theta^{i)}$ as input and $Eul[\Omega^{0)}(\Omega^{i)})^T]$ as output
Learn τ with $X^{i)} - \Delta X(\Omega^{i)})$ as input and $\theta^{i)}$ as output

3.3 Fully overlapped (on-line) learning

None of the above learning strategies can be used to perform on-line learning, i.e., learning that is integrated in normal working operation. The strategy that we present now parallelizes the learning of all the functions used in our procedure to calculate the IKM. And, interestingly, it permits carrying out arbitrary movements, as for example those required by an application, while at the same time refining the estimation of the IKM of the robot.

To get these advantages we need access to the inverse $\phi_0^{-1}(\nu)$, which gives the orientation that the argument ν produces when $\theta = \theta_0$. Fortunately, being the inverse of ϕ_0, it can be learned from its same input-output samples. In some types of systems (Walter and Ritter, 1996), even, the learning of a function automatically makes available a proper estimation of its inverse and, therefore, a separate estimator for $\phi_0^{-1}(\nu)$ would not be required. This is the algorithm:

Learning of Ω_0, τ, ϕ_0 and ϕ_0^{-1}
Select θ_0 and $\nu^{0)}$ arbitrarily
Move to $(\theta_0, \nu^{0)})$. Observe $\Omega^{0)}$
Repeat for $i = 1$ to whatever
Select $(\theta^{i)}, \nu^{i)})$
Move to $(\theta^{i)}, \nu^{i)})$. Observe $(X^{i)}, \Omega^{i)})$
Learn Ω_0 with $\theta^{i)}$ as input and $Eul[\phi_0^{-1}(\nu^{i)})\ (\Omega^{i)})^T]$ as output
Learn τ with $X^{i)} - \Delta X(\Omega^{i)})$ as input and $\theta^{i)}$ as output
Learn ϕ_0 with $Eul[\Omega_0(\theta^{i)})\ (\Omega^{i)})^T]$ as input and $\nu^{i)}$ as output
Learn ϕ_0^{-1} with $\nu^{i)}$ as input and $Eul[\Omega_0(\theta^{i)})(\Omega^{i)})^T]$ as output

In this algorithm, τ is learned without any help of the other functions. On the contrary, the learning of ϕ_0 (and its inverse) is supported by Ω_0 and, reciprocally, ϕ_0 is needed to learn Ω_0. dependence is Three entities should be clearly distinguished: 1) the actual functions Ω_0, ϕ_0 and ϕ_0^{-1} resulting from the robot geometry, 2) the estimations of these functions, and 3) the data generated to estimate the functions. There is a positive feedback between Ω_0 and ϕ_0: when the estimator of Ω_0 ameliorates, the data provided to learn ϕ_0 is more exact, as well as ϕ_0 itself, which redounds to the accuracy of the data for Ω_0.

But feedback can also be negative. Imagine for example that our learning system has arrived to the desired accuracy in the calculus of the IKM and a serious and sudden damage happens, affecting only the last elements of the robot. Ω_0 will remain accurate, but ϕ_0 (and its inverse) will not. In this mode of learning, the inaccuracy of Ω_0 will lead to erroneous data for ϕ_0 (and its inverse), which will worsen its estimation. Thus, after a sudden and important damage, it could be wise to check whether Ω_0 is accurate and, if it is so, suppress the step where it is learned. It is important to note that in the case of damage to the first links, not only τ and Ω_0 will be affected, but usually also ϕ_0 (and its inverse), since the reference θ_0 will not produce in general the same orientation with any given ν as before damage. Therefore, any checking of ϕ_0 (or its inverse) alone is useless.

In a certain sense, Ω_0 and ϕ_0 are learned exclusively from one another: the error in the training data for Ω_0 is exactly the approximation error for ϕ_0^{-1}, and the error in the training data for ϕ_0^{-1} is exactly the approximation error for Ω_0. So, how can the learning of these functions progress? It suffices to have a starting point in the form of a known point for one of the functions. Such a point to initiate positive feedback is always available for Ω_0: $(\theta_0, 0_R)$, where 0_R stands for the null rotation representation. For memory-based systems it is enough to provide it as a first point and keep it unaltered if necessary. For systems requiring repeated presentations of the training patterns, a periodical remind is convenient.

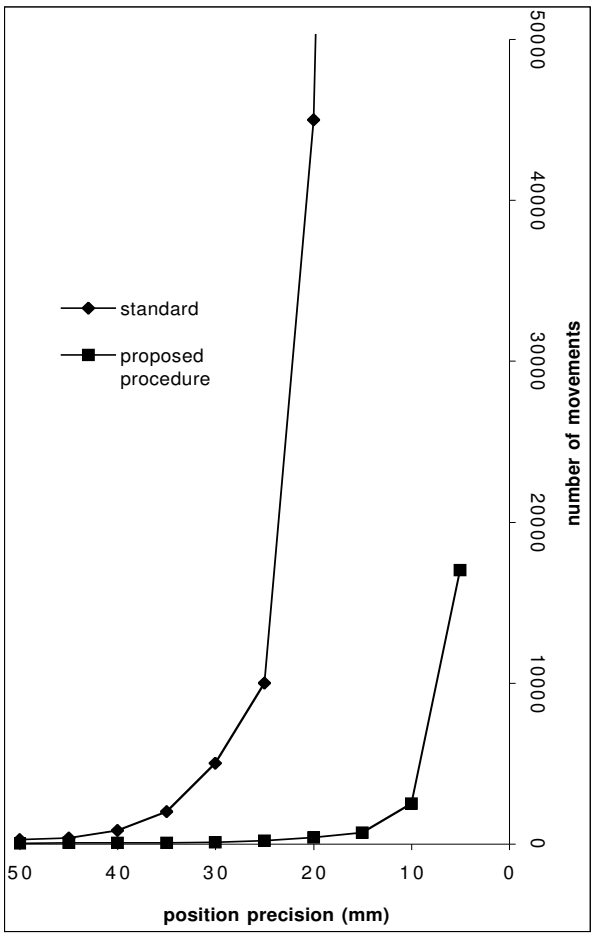

Figure 1: Number of movements required to obtain different levels of position precision.

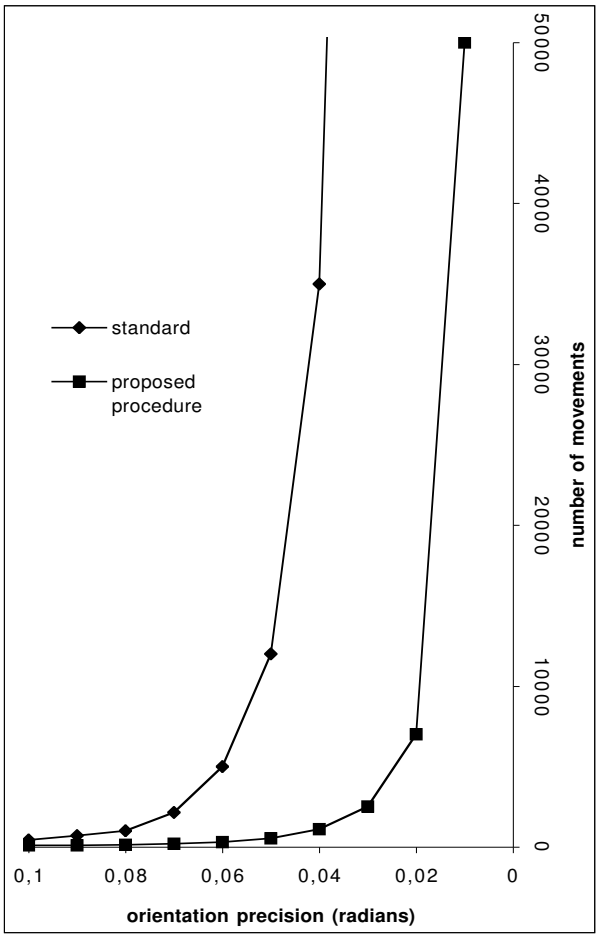

Figure 2: Number of movements required to obtain different levels of orientation precision.

4 Some preliminary results

We have not yet experimented this decomposition approach systematically with sophisticated learning systems, but we have made some preliminary tests with two simple systems, namely back-propagation networks and the nearest neighbor (NN) algorithm. We present results with the latter, since comparisons with the former are more prone to subjectivity due to variable factors, such as architecture, learning algorithm and degree of training, that cannot be optimized with the same values for the two experiments we want to conduct to compare results.

We have chosen the PUMA robot for our simulations to validate our procedure in a controlled setting. The workspace used was generated by a range of 30 degrees in each of the six joints.

For the control experiment (labeled "standard") we simply generate random movements of the robot in the range above and observe the resulting positions and orientations. These and their associated joint configurations are added to the training set after appropriate normalization. When the NN system is queried with a desired position and orientation, it searches the closest position-orientation vector stored and returns as output the corresponding joint values.

In the other experiment, we tested our partially overlapping procedure: the robot was moved with a random ν to get a point for ϕ_0, and again with random θ to get points for τ and Ω_0 in each iteration. The learners for the three functions were NN systems analogous to the one used in the first experiment.

Figures 1 and 2 show the total number of movements required to get different precision levels. Units are millimetres for position and radians for orientation. Standard deviations are below the resolution level of the graphic and, thus, are not shown. The precision was evaluated by querying for 200 random position-orientation configurations in-

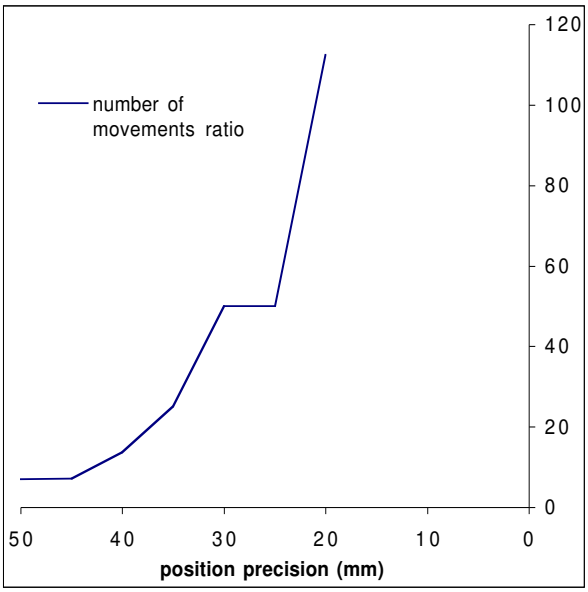

Figure 3: Ratio between the two curves displayed in Figure 1.

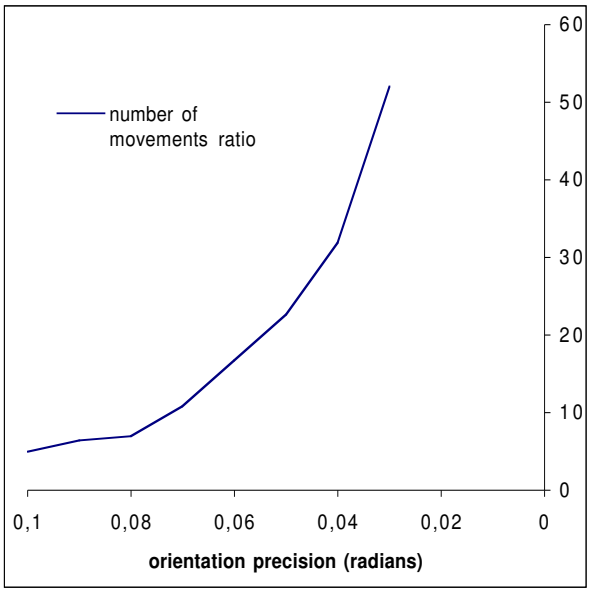

Figure 4: Ratio between the two curves displayed in Figure 2.

side the workspace. It is interesting to detail some of the data used to build the graphic: while the standard procedure needed 280 movements to attain a precision of 50 mm and 440 movements to attain a precision of .1 radians, the partially overlapping procedure only needed 40 and 90 movements respectively to obtain the same precisions. Moreover, higher precisions enlarge the differences: the standard procedure required 45000 and 35000 movements to get precisions of 20 mm and .4 radians respectively, whereas the partially overlapping procedure only needed 400 and 1100 movements respectively.

To reveal more clearly the benefit of the partially overlapping procedure with increasing precisions, we have displayed in Figures 3 and 4 the ratios between the number of movements required by the standard and the partially overlapping procedures. A polynomical scaling of this ratio with the precision required is appreciable both for position and orientation.

5 Concluding remarks

The purpose of this paper is to propose a method to learn the IK mapping with a reasonable number of movements when a high accuracy is required.

To this end, we assume that the axes of the last three joints cross at a point, which is a condition fulfilled by most robot arms. This condition holds after the most likely miscalibrations such as, for example, encoder shift. More severe physical damage affecting the first links (those whose axes are not required to cross) is also allowed. Even the gripper or a physical element linking the cross point to the gripper can also be deformed without violating the assumption.

One of the most promising applications of our method is the learning of IK for flexible robots. Usually the first links are much longer and heavier than the last ones, which are used mainly to give an appropriate orientation to the gripper. This makes the first links more prone to elastic deformation due to lever effect. If the last links are short and robust, the cross condition is valid for this type of robot. Since our method reduces the dimensionality of the functions to be learned from 6 to 3, it is still affordable to include the weight changes as an extra variable and still have quick learning (as a matter of fact, one only needs to add this variable as an input in the learning of Ω_0 and τ).

In addition to learning efficiency, our method has other advantages over classic learning of IK in some contexts. For example, in (Ruiz de Angulo and Torras, 1997) we tackled the learning of a space robot whose mission was to insert and extract cards from a rack. If, due to launching stress or tear-and-wear, the IK mapping strongly deviates from the stored one, the movements required for relearning could damage the rack (or further damage the robot). With the method here proposed, it is possible to learn to move in the complete workspace without

actually moving everywhere, and only approach risk zones after learning is completed.

References

Kröse B.J.A. and van der Smagt P.P., 1993: *An Introduction to Neural Networks* (5th edition), Chapter 7: "Robot Control". University of Amsterdam.

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

Ritter H., Martinetz T. and Schulten K.J. , 1992: *Neural Computation and Self-Organizing Maps*. New York: Addison Wesley.

Ruiz de Angulo V. and Torras C., 1997: Self-calibration of a space robot. *IEEE Trans. on Neural Networks*, **8(4)**:951-963.

Walter J.A. and Schulten K.J., 1993: Implementation of self-organizing neural networks for visuo-motor control of an industrial arm. *IEEE Trans. on Neural Networks*, **4(1)**.

Walter J. and Ritter H., 1996: Rapid learning with parametrized self-organizing maps. *Neurocomputing*, **12**:131-153.