Speeding up the learning of robot kinematics through function decomposition

Vicente Ruiz de Angulo and Carme Torras

Corresponding author's e-mail: torras@iri.upc.edu

Abstract

The main drawback of using neural networks or other example-based learning procedures to approximate the inverse kinematics (IK) of robot arms is the high number of training samples (i.e., robot movements) required to attain an acceptable precision. We propose here a trick, valid for most industrial robots, that greatly reduces the number of movements needed to learn or relearn the IK to a given accuracy. This trick consists in expressing the IK as a composition of learnable functions, each having half the dimensionality of the original mapping. Off-line and on-line training schemes to learn these component functions are also proposed. Experimental results obtained by using Nearest Neighbours and PSOMs, with and without the decomposition, show that the time savings granted by the proposed scheme grow polynomially with the precision required.

This paper is an extended and updated version of [7], presented at the International Conference on Artificial Neural Networks (ICANN-02).

The authors are with the Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Parc Tecnològic de Barcelona - Edifici U, Llorens i Artigues 4-6, 08028-Barcelona, Spain. E-mail: {ruiz,torras}@iri.upc.edu ; http://www-iri.upc.es

Index Terms

Learning inverse kinematics, PSOMs, training samples, function approximation, robot kinematics.

I. INTRODUCTION

A robot manipulator is a multifunctional and reprogrammable articulated mechanism able to move in a given workspace. It usually consists of several bodies linked by joints, and it is commanded by providing values to some of these joints. Thus, when moving, the robot can be thought of as realizing a mapping from joint space to workspace coordinates, which is referred to as the *forward kinematics mapping*. Robot programming, however, is most easily carried out in terms of the cartesian coordinates of the workspace, leaving to the controller the task of translating such specification into joint variables. Thus, robot control critically depends on the so-called *inverse kinematics mapping* (IKM), i.e. that providing joint coordinates as a function of the desired position and orientation of the robot end-effector in the workspace.

Their range of application would widen if robots were made adaptive not only to environmental variations, but also to changes in their own geometry. Since these geometric changes affect the IKM, a way of learning (or tuning) this mapping automatically while robots move is highly desirable. Recently, the development of humanoid robots has further raised the interest in this problem [1]. An overview of the approaches proposed to learn the IKM is provided in [9].

Such learning is especially useful for robots that have 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. Moreover, rigid non-redundant robots also benefit from such learning, since this permits their on-line recalibration during normal functioning. In particular, high-precision robots may need to be recalibrated often, which makes

some applications impractical or impossible without such learning capability. 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 the requirement of a sophisticated set-up, with sensors able to determine the position and orientation of the end-effector. This set-up makes the system more expensive, but may also be useful for other purposes. A more serious drawback is the high number of samples often required to approximate the mapping up to the desired accuracy. This paper proposes a solution to this second drawback.

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 powerfully influencing 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 [3], [4].

Several attempts have been made at reducing the number of required samples, among them the use of hierarchical networks [5], [11], the learning of only the deviations from the nominal kinematics [6], and the use of a continuous representation by associating a basis function to each node [10].

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 and much simpler learning tasks. This is done at the expense of sacrificing generality: the procedure works only for some robot models subject to certain types of deformations. Specifically, the procedure assumes that the last three robot joints cross at a point. This is fulfilled by the most 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 input dimensionality of each of the tasks resulting from the decomposition is half that of the original one. Thus, for a given desired accuracy, if the number of training samples required to learn the IKM directly is $O(n^d)$, through the decomposition it reduces to $O(n^{d/2})$. This yields an enormous reduction in the number of samples 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 the training scheme needed to learn the component functions. In Section 4, both a nearest-neighbour algorithm and a parameterized self-organizing map (PSOM) are used to learn the IKM, both directly and through the decomposition, permitting to quantify the savings obtained in relation to the precision required. Finally, some conclusions are drawn in Section 5.

II. Decomposing the inverse kinematics mapping

As mentioned in the preceding section, the number of samples required to learn the IKM grows exponentially with the number of input variables. To keep this growth within reasonable bounds, we propose to decompose the IKM in such a way that each component function depends on only half of the input variables. Since the input, in this case, is naturally divided into position and orientation, every component function should depend on either the desired position or orientation

alone. The most interesting case of robot with six rotational joints for which such decomposition is possible is that in which the axes of the three last joints cross at a point. We will formulate the decomposition for this case, the most common one, although it can be applied to other types of robots as well.

A. Overview of the proposed approach

In order to make the formal derivation of our approach more understandable, we first provide the reader with an intuitive view of what we are aiming at. Figure 1 shows the involved variables and mappings particularized for the widely-known PUMA robot. $\theta = (\theta_1, \theta_2, \theta_3)$ and $\nu = (\nu_1, \nu_2, \nu_3)$ are the values of the first three and the last three joint angles, respectively. X and Ω are the desired position and orientation of the end-effector. Then, the full IKM maps (X, Ω) into (θ, ν). Our proposed decomposition consists of two mappings (labelled "translation" and "rotation" in the figure) yielding θ and ν, whose inputs are X and Ω, respectively, combined with apropriate offset functions. The meaning of the offsets will become clear in what follows. Here we just like to point out that, in this case, the two inverse mappings and the two offsets are 3D functions, while the full IKM is a 6D function.

In what follows, 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.

B. 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 depend only on either X or Ω.

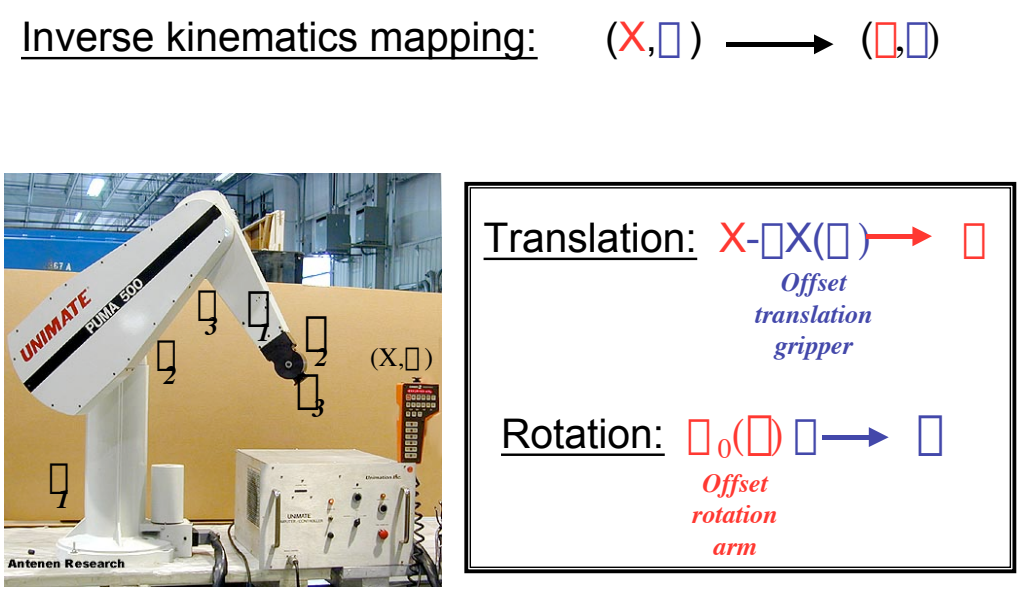

Fig. 1. The photograph shows a PUMA robot with the variables $\theta = (\theta_1, \theta_2, \theta_3)$ and $\nu = (\nu_1, \nu_2, \nu_3)$ for the first three and the last three joint angles, respectively, superimposed. X and Ω are the desired position and orientation of the end-effector. On top, the original, full 6-to-6 inverse kinematic mapping, is specified; and below on the right, the proposed decomposition of the mapping is sketched. Its two components are 3-to-3 inverse mappings, depending on only translation and rotation, respectively, plus off-sets as described in the text.

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. This $\Delta X(.)$ function is the translation offset mentioned in Fig. 1.

Thus, $\tau : X^* \rightarrow \theta$ is a partial inverse kinematics mapping (the translation component

mentioned in Fig. 1) with reduced dimensionality with respect to the original problem. It may be multivalued for unrestricted workspaces, and it can be learned with known methods. Since X^* is not directly available, it must be previously calculated:

$$\theta = \tau(X^*) = \tau(X - \Delta X(\Omega)). \tag{2}$$

The correctness of X^* guarantees the correctness of X conditioned on the correctness of Ω. Therefore, we shall guarantee the correctness of Ω by using the remaining degrees of freedom, ν, as shown below.

C. Calculus of ν

To calculate ν, we consider a simplified version of the inverse kinematics mapping by freezing θ in a reference configuration. In this way, Ω only depends on ν, thus making $\Omega \rightarrow \nu$ a learnable inverse kinematics mapping, namely the rotation component in Fig. 1. We need to model the relation between Ω_1 in an arbitrary θ_1 configuration with the orientation in a reference configuration θ_0, which will provide us with the rotation offset mentioned in Fig. 1. Let us write this down formally. To simplify the exposition, we assume that Ω is represented as 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):

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

Figure 2 illustrates this function. Note that $\Omega_0(.)$ is independent of ν and the only requirement is that the last links and joints in ν are not flexible.

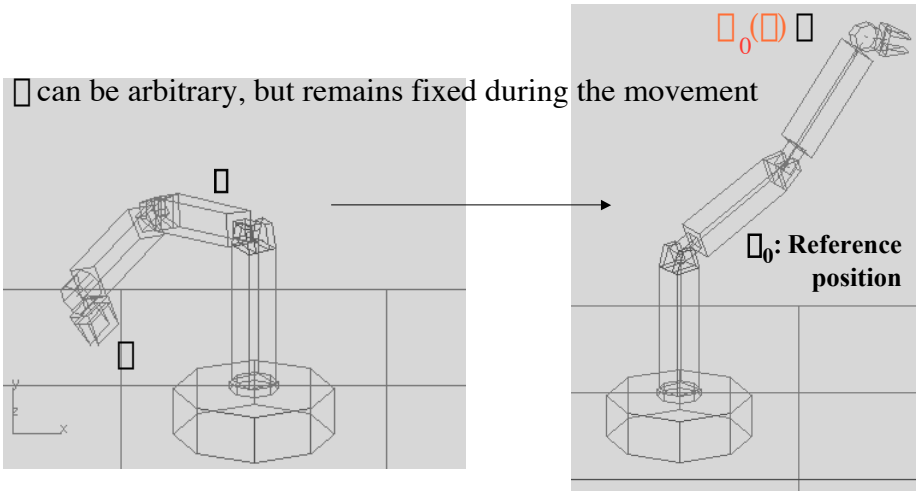

Fig. 2. Illustration of the rotation offset function $\Omega_0(\theta)$. When the robot moves the first three joints from θ (left) to θ_0 (right) while maintaining the last three joints fixed, the gripper undergoes a rotation of $\Omega_0(\theta)$ going from Ω (left) to $\Omega_0(\theta)\Omega$ (right).

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.

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,

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

$\phi_0(\cdot)$ is a 3-to-3 inverse mapping, which constitutes the rotation component mentioned in Fig. 1.

D. The target decomposition

Table I summarizes the four functions involved in the 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. The first diagram in the appendix illustrates this two-phase process.

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. 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.

Function	Description
$\tau : X^* \rightarrow \theta$	**Inverse kinematics (only position)** of the cross-point, the point where the last three axes cross
$\Delta X : \Omega \rightarrow T$	**Offset for τ:** Translation of the gripper position X returning the cross-point position X^*
$\phi_0 : \Omega \rightarrow \nu$	**Inverse kinematics (only orientation)** of the gripper when θ is fixed to θ_0
$\Omega_0 : \theta \rightarrow R$	**Offset for ϕ_0:** Rotation undergone by the gripper when θ changes while ν remains fixed

TABLE I
THE FOUR FUNCTIONS INVOLVED IN THE DECOMPOSITION OF THE INVERSE KINEMATICS LEARNING.

III. 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 X^*. Thus, the total number of observations required to derive X^* from X 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 encoding 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.

To help visualize the data flow for the different learning strategies, flow diagrams for each of them are included in the appendix.

A. 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 $\Omega^{0)}(\Omega^{i)})^T$ 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 $\Omega^{i)}$ as input and $\nu^{i)}$ as output

B. Partially overlapped learning

There are alternatives more efficient than the independent learning of all functions. We first suggest the parallelization of the encoding of ΔX (if X^* cannot be directly acquired by means of external sensors) 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 $\Omega^{0)}(\Omega^{i)})^T$ as output

Learn τ with $X^{i)} - \Delta X(\Omega^{i)})$ as input and $\theta^{i)}$ as output

C. 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 [10], 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}

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 $\phi_0^{-1}(\nu^{i)})\ (\Omega^{i)})^T$ as output

Learn τ with $X^{i)} - \Delta X(\Omega^{i)})$ as input and $\theta^{i)}$ as output

Learn ϕ_0 with $\Omega_0(\theta^{i)})\ \Omega^{i)}$ as input and $\nu^{i)}$ as output

Learn ϕ_0^{-1} with $\nu^{i)}$ as input and $\Omega_0(\theta^{i)})\Omega^{i)}$ as output

When this algorithm is integrated in the normal operation of the robot, $\theta^{i)}$ and $\nu^{i)}$ are the configurations where the robot is moved to, instead of being generated randomly. 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. 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.

IV. Experimental results

We have used the PUMA robot as a testbed to validate our procedure in a controlled setting. Three learning systems have been tested with our decomposition approach, namely backpropagation networks, the nearest neighbor algorithm, and local parametrized self-organizing maps (PSOM) [10]. We present results with the last two ones, since comparisons with the former are more prone to subjectivity due to variable factors, such as architecture, learning algorithm and degree of training, which cannot be optimized with the same values for the two experiments that need to be conducted to compare results.

A. Results using the nearest-neighbour algorithm

The workspace used was generated by allowing 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 nearest-neighbour algorithm 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 test our partially overlapping procedure: the robot is 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 are nearest-neighbour algorithms analogous to the one used in the first experiment.

Orientations and rotations are represented with five elements (last column and last row) of the corresponding rotation matrix that determine it univocally except in gimbal lock situations.

Figures 3 and 4 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 inside 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 5 and 6 the ratios between the number of movements

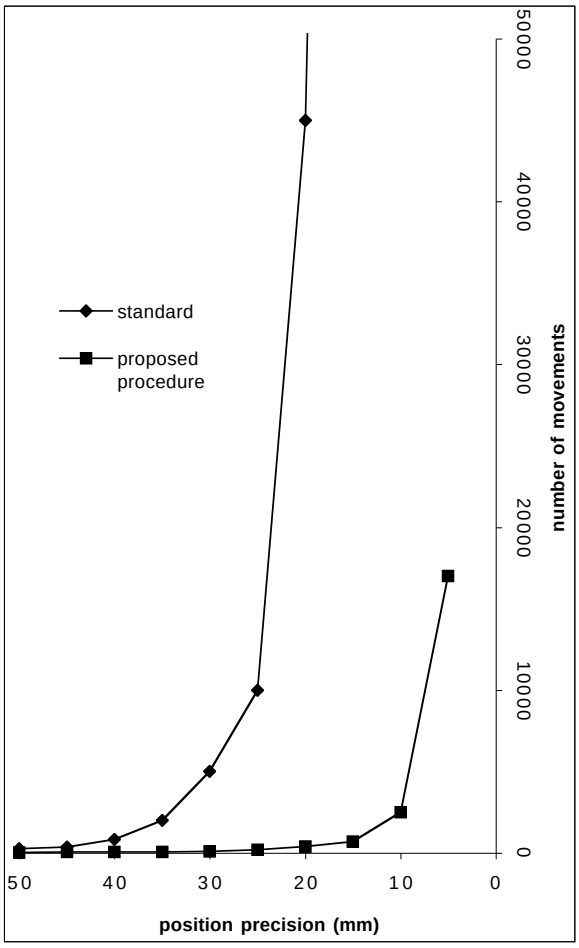

Fig. 3. Number of movements required to obtain different levels of position precision using the nearest neighbour algorithm.

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.

B. Results using Local PSOM's

A Parametrized Self-Organized Map (PSOM) [10] approximates a function using a regular grid of sampled points, the nodes of the network. Because of its excellent interpolation capabilities, the required number of points is very small. Of particular interest to us is that PSOMs treat input and output variables in the same way. This means that it is as natural to ask which output

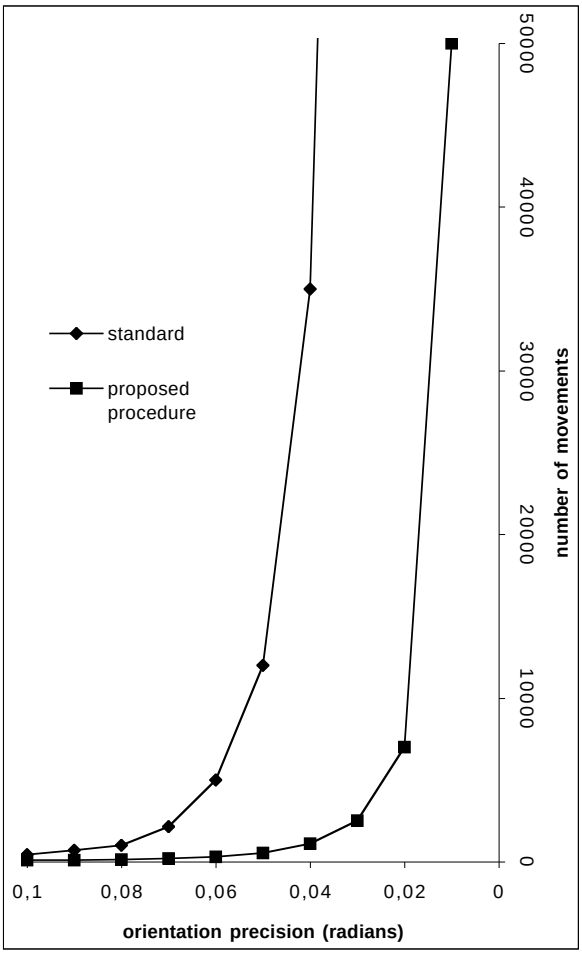

Fig. 4. Number of movements required to obtain different levels of orientation precision using the nearest neighbour algorithm.

corresponds to a given input as asking which input correspond to a given output. Therefore, a search in the input variables is naturally addressed and embedded in the framework of these networks, allowing to manage inverse-multivalued functions without problems.

When using PSOMs to learn the kinematics of the virtual robots, the movements are generated following a regular grid in the space of joint angles covering the workspace. Then we move the robot to the different configurations represented in the grid to obtain the associated positions and orientations. Thus, each node in the grid requires one movement. Once trained, a PSOM works by putting some constraints on a subset of the variables of the system (input or output),

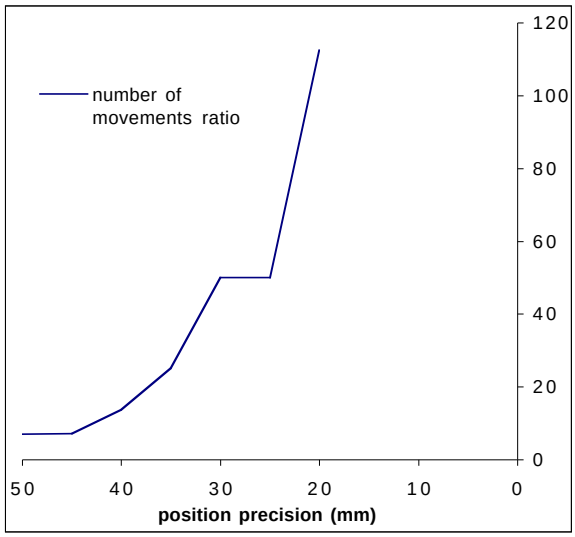

Fig. 5. Ratio between the two curves displayed in Figure 3.

for example fixing them to a desired value. The system then carries out a quick optimization aimed at finding a point of the approximated input-output manifold satisfying the constraints or, if impossible, the closest one to satisfying them. The starting point of the process is the stored point (node) that best satisfies the constraints. From it, an iterative minimization procedure is launched, which finishes in a few steps. For PSOMs trained on the kinematics of a robot, to get the inverse kinematics we simply fix the position and orientation variables, and we let the minimization get the point in the interpolating surface with the desired pose values, then the remaining components of the point are taken to be the result. This is the control (”standard”) experiment.

In the experiment to test our decomposition approach, a PSOM is created for each of the functions to be learned: we generate a grid for θ and move the first three robot joints to traverse each of its points in order to get simultaneously points for τ and Ω_0. In the same way, a grid for ν is generated and movements are carried out accordingly to get points for ϕ_0. This corresponds

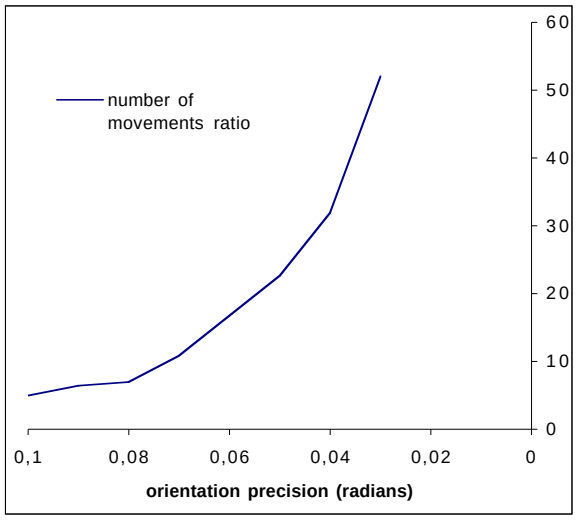

Fig. 6. Ratio between the two curves displayed in Figure 4.

to the partially overlapping learning procedure of Section III.B. In the operation phase, to get for example the value of ϕ_0, we simply fix the orientation values of the corresponding PSOM.

In the experiments presented in this section we used a PSOM variant known as LPSOM [10]. The "L" stands for "Local", because this model builds a PSOM by extracting for each query a subgrid of the sampling grid, which is centered on the closest point to the query. This subgrid has a size of 4 points per axis in our tests.

For the bunch of experiments carried out with PSOMs, the workspace for the PUMA robot has been considerably enlarged. The ranges allowed for the six joints [2] in these experiments are as follows: $[-150, -10]$, $[-215, -100]$, $[-35, 80]$, $[-110, 170]$, $[-100, 100]$, $[-100, 100]$.

Orientations and rotations are represented as before with five elements of the corresponding rotation matrix.

Tables II and III show the precisions attained with an increasing number of movements. Note that since the movements in Table II correspond to PSOM grids of dimension 6, while those

in Table III are obtained with two grids of dimension 3, it was not possible to have the same number of movements in both tables (n^6 in the former, versus $2n^3$ in the latter). The precision was evaluated by querying for 400 random position-orientation configurations inside the workspace. The tables only cover numbers of movements that seem reasonable. It was impossible with our computer memory resources (allowing grids of up to 262,144 points) to reach precisions under 1 mm and .01 radians with the standard procedure, whereas the decomposition procedure only needed 686 and 1024 movements to get these precisions, respectively.

A final and important remark is that the time to obtain good precisions was also orders of magnitude faster with the decomposition approach. This is due to lower searching times to get the closer node in the grids and to lower complexity in the optimizations performed in the PSOMs.

number of movements	position mean error	position stdev. error	orientation mean error	orientation stdev. error
64	591	234	2.145	0.655
729	195	132	0.316	0.253
4096	38	50	0.138	0.163

TABLE II
POSITION (IN MILLIMETERS) AND ORIENTATION (IN RADIANS) PRECISIONS OBTAINED WITH DIFFERENT NUMBERS OF MOVEMENTS USING THE STANDARD PROCEDURE.

V. CONCLUDING REMARKS AND FUTURE WORK

The purpose of this paper is to propose a procedure to learn the Inverse Kinematics (IK) mapping with a reasonable number of movements when a high accuracy is required.

number of movements	position mean error	position stdev. error	orientation mean error	orientation stdev. error
54	57.7	27.2	0.420	0.255
128	9.7	6.0	0.158	0.150
250	3.0	2.7	0.068	0.134
432	1.0	0.8	0.020	0.029
686	0.6	1.0	0.015	0.032
1024	0.2	0.2	0.006	0.028

TABLE III
POSITION (IN MILLIMETERS) AND ORIENTATION (IN RADIANS) PRECISIONS OBTAINED WITH DIFFERENT NUMBERS OF MOVEMENTS USING THE NEW DECOMPOSITION PROCEDURE.

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 robots. 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 [6] we tackled IK learning for a REIS robot placed in a Space Station mock-up, whose mission was to insert and extract cards from a rack. If, due to launching stress or tear-and-wear, the IK mapping would strongly deviate from the nominal one, the movements required for relearning could damage the rack (or further damage the robot). With the procedure here proposed, it is possible to learn to move in the complete workspace without actually moving everywhere, and only approach risk zones after learning has been succesfully completed.

A possible way of improving orientation accuracy is to change the representation of rotations and orientations in the learning systems. We have used five elements of the rotation matrix, which guarantees that rotations that are close to one another have also close representations (a property not exhibited, for example, by Euler angles and quaternions). This is not the best representation for learning since, for example, the full rotation matrix leads to lower errors, although it is more expensive in computation and memory and, overall, has the problem of how to map the interpolated matrices to true rotation matrices. This representation issue is not particularly linked to our decomposition procedure. It can be avoided in future work by calculating ν as function of positions and translations instead of orientations and rotations.

We have to mention that a very similar decomposition procedure can be developed for robots not fulfilling the cross-point condition, but whose first three joints are prismatic. A more involved task is the development of a general decomposition procedure for serial manipulators with arbitrary joints. This procedure cannot be obtained as a straightforward generalization of the one presented in this paper, since here we have exploited the condition that the last three joint

axes cross at a point. Some preliminary work in this direction can be found in [8].

To conclude, let us stress that our decomposition procedure places requirements on the robot to which it is applied, but not on the learning method and, thus, it can be used in combination with any such method based on input-output training samples.

APPENDIX

A. *Operation module of the proposed decomposition approach.*

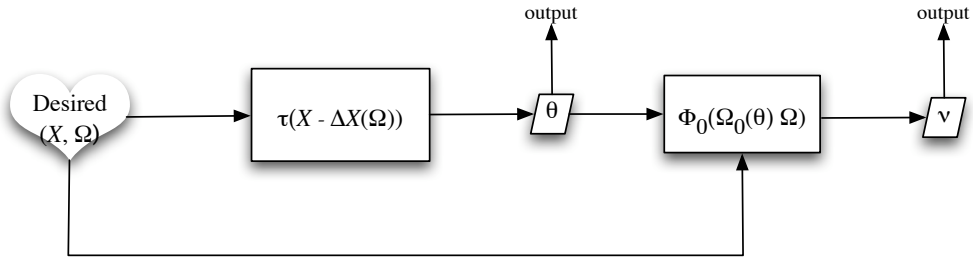

Fig. 7. Flow-diagram of the proposed approach in operation. It shows how the inverse kinematics of the robot is calculated in two stages.

B. Independent learning strategy.

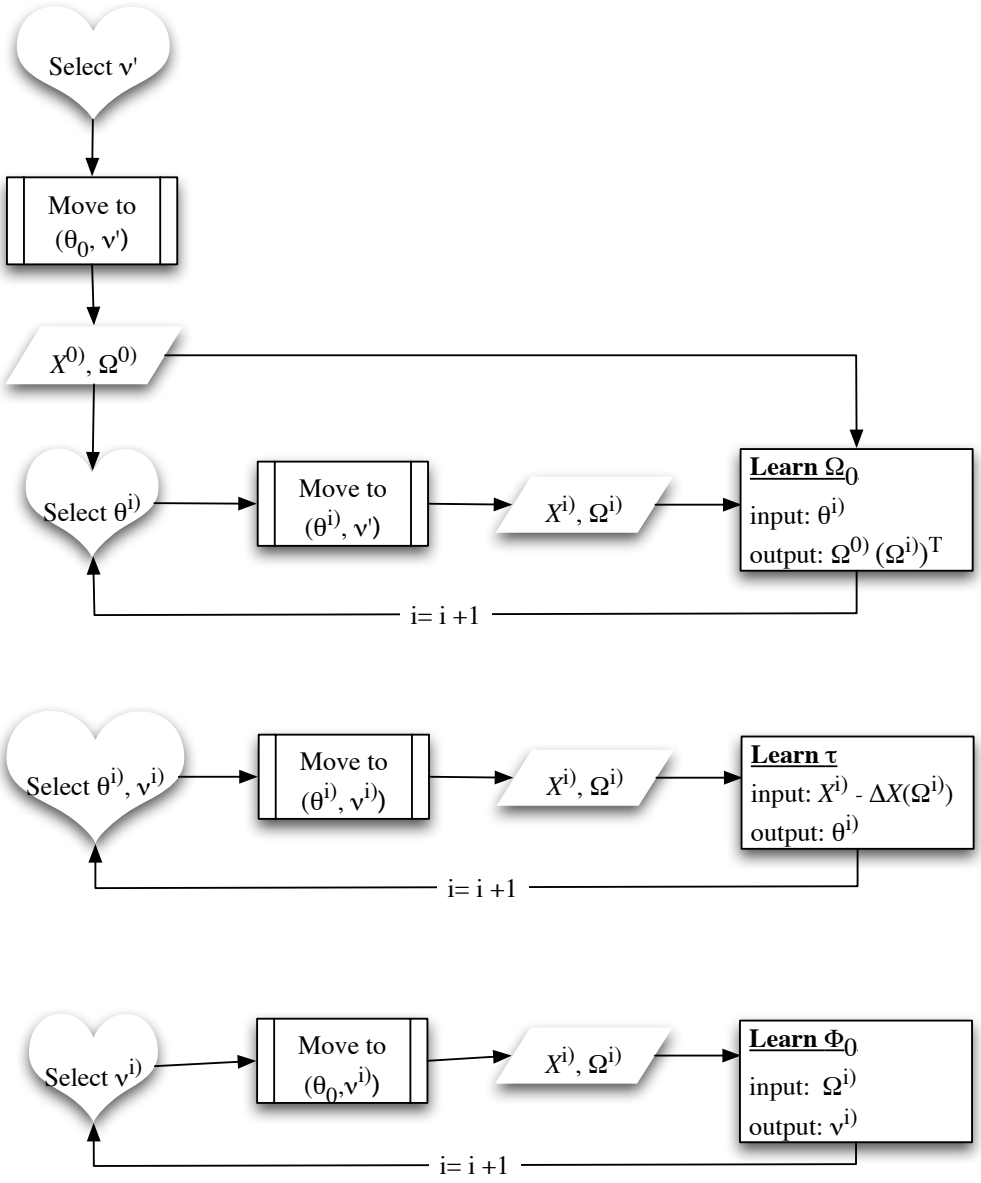

Fig. 8. Flow-diagram for the independent learning of $\Omega_0(\cdot)$, $\tau(\cdot)$ and $\phi_0(\cdot)$.

C. Partially overlapped learning strategy.

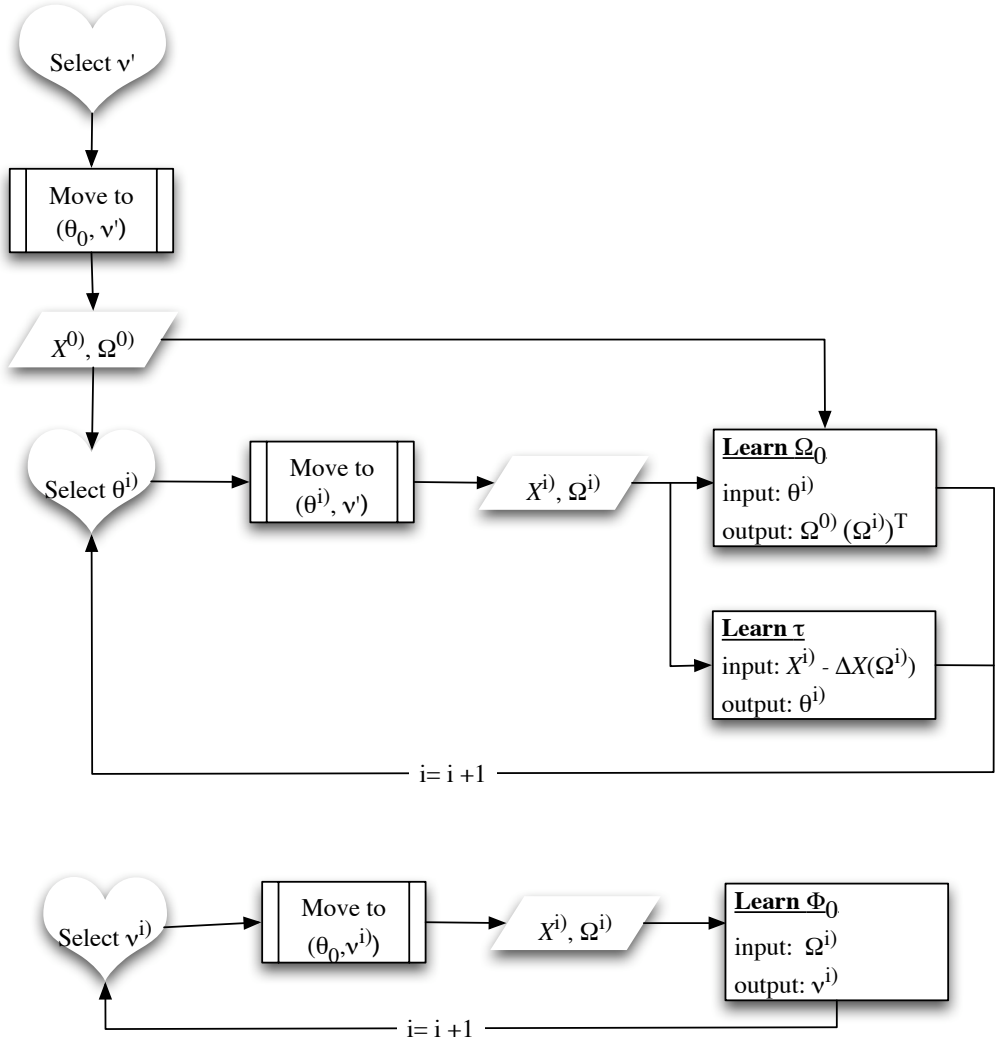

Fig. 9. Flow-diagram for the overlapped learning of $\Omega_0(\cdot)$ and $\tau(\cdot)$, while $\phi_0(\cdot)$ remains being learned independently.

D. Fully overlapped learning strategy.

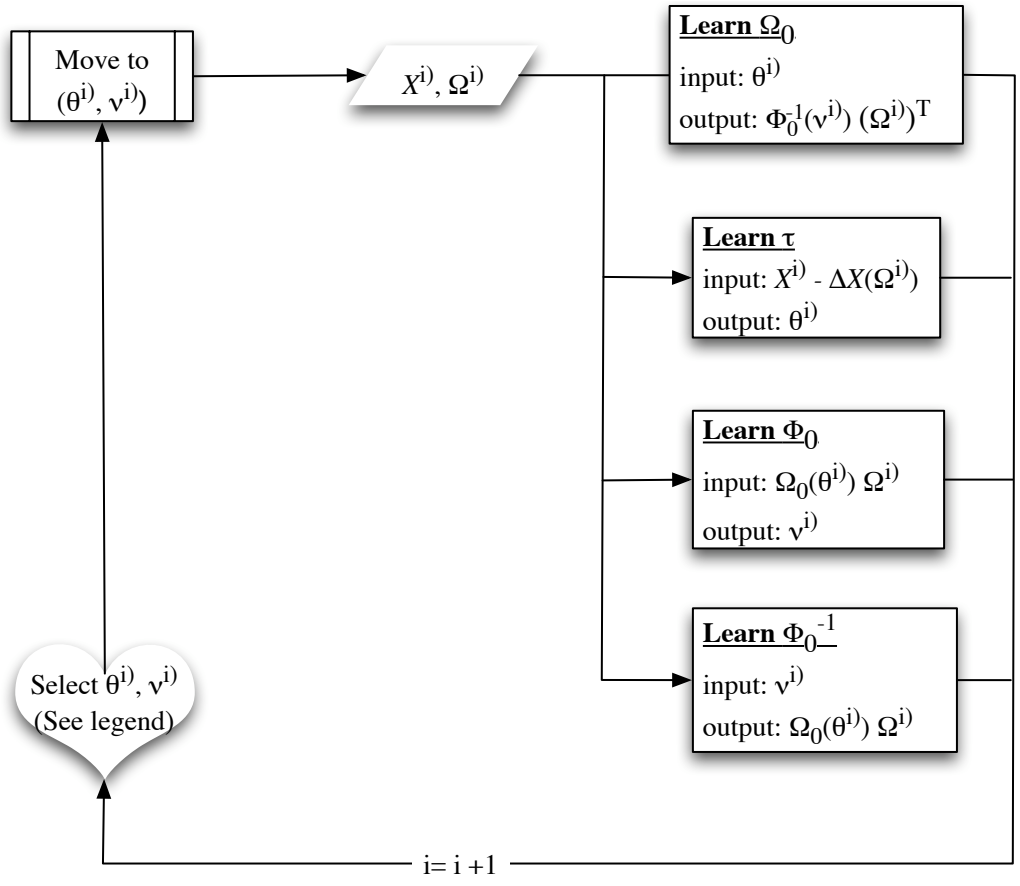

Fig. 10. Flow-diagram for the fully overlapped learning of $\Omega_0(\cdot)$, $\tau(\cdot)$, $\phi_0(\cdot)$ and $\phi_0^{-1}(\cdot)$. The selection of $\theta^{i)}$ and $\nu^{i)}$ can be done either directly or through the **operation module**.

ACKNOWLEDGEMENT

This research has been partially supported by the Catalan Research Commission, through the Robotics and Control group, and by the I+D project DPI 2004–07358 of the Spanish Ministry of Education.

REFERENCES

[1] A. D'Souza, S. Vijayakumar and S. Schaal, Learning inverse kinematics, Proc. IEEE/RSJ Conf. on Intel. Robots and Systems (IROS'01), Hawaii, USA, pp. 298-303, 2001.

[2] K.S. Fu, R.C. González and C.S.G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, New York: McGraw-Hill, 1987.

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

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

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

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

[7] V. Ruiz de Angulo and C. Torras, Learning inverse kinematics via cross-point function decomposition. Proc. Intl. Conf. on Artificial Neural Networks (ICANN-02), *Lecture Notes in Computer Science* 2415: 856-861, 2002.

[8] V. Ruiz de Angulo and C. Torras, Using PSOMs to learn inverse kinematics through virtual decomposition of the robot, Proc. 8th Intl. Work-Conference on Artificial Neural Networks (IWANN'2005), *Lecture Notes in Computer Science*, 2005.

[9] C. Torras, Robot arm control, in *Handbook of Brain Theory and Neural Networks - 2nd edition*, edited by M.A. Arbib, pp. 979-983, MIT Press: Cambridge, Massachusetts, 2003.

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

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

Vicente Ruiz de Angulo was born in Miranda de Ebro, Burgos, Spain. He received the B.Sc. and the Ph.D. degrees in computer science from the Universidad del País Vasco. During the academic year 1988-89, he was Assistent Professor at the Universitat Politècnica de Catalunya. In 1990 he joined the Neural Network Laboratory of the Joint Research Centre that the Commission of the European Union has in Ispra (Italy). From 1995 to 1996 he was with the Institut de Cibernètica, in Barcelona, participating in the ESPRIT project entitled "Robot Control Based on Neural Network Systems" (CONNY). He also spent six months at the Istituto Dalle Molle di Studi Sull' Inteligenza Artificiale (IDSIA) di Lugano, working in applications of neural networks to robotics. Since 1996 he is with the Institut de Robòtica i Informàtica Industrial in Barcelona. His interests in neural networks include fault tolerance, noisy and missing data processing and their application to robotics and computer vision.

Carme Torras (http://www-iri.upc.es/people/torras) is research professor at the Institut de Robòtica i Informàtica Industrial (CSIC-UPC). She received M.Sc. degrees in Mathematics and Computer Science from the Universitat de Barcelona and the University of Massachusetts, respectively, and a Ph.D. degree in Computer Science from the Universitat Politècnica de Catalunya. Prof. Torras has published four books and more than a hundred papers in the areas of neurocomputing, robotics and vision. She has been local project leader of several European projects, such as "Robot Control based on Neural Network Systems" (CONNY), "Self-organization and Analogical Modelling using Subsymbolic Computing" (SUBSYM), "Planning RObot Motion" (PROMotion) and "Behavioural Learning: Sensing and Acting" (B-LEARN).