Learning inverse kinematics via cross-point function decomposition

Vicente Ruiz de Angulo and Carme Torras

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

Abstract. The main drawback of using neural networks 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. A training scheme to learn these component functions is also proposed. Experimental results obtained by using PSOMs, with and without the decomposition, show that the time savings granted by the proposed scheme grow polynomically with the precision required.

1 Introduction

Making robots adaptive not only to environmental conditions, but also to changes in their own geometry, would widen their range of application. These geometric changes affect the Inverse Kinematics Mapping (IKM) relating workspace coordinates (in which tasks are usually specified) to joint coordinates (used to command the robot). The interest of learning this mapping in the case of flexible or redundant robots is widely recognized. Moreover, rigid non-redundant robots would also benefit from such learning, since this would permit their on-line recalibration during normal functioning.

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, rather than to attain an accurate approximation, 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 training points required may be huge [3, 2].

Several attempts have been made at reducing the number of required samples, among them the use of hierarchical networks [4, 6], the learning of only the deviations from the nominal kinematics [5], and the use of a continuous representation by associating a basis function to each neuron [7].

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 trick works only for robots whose last three joints cross at a point. This condition is fulfilled by the most popular robot arms, and 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, if the number of training points 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 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 the training scheme needed to learn the component functions. In Section 4, a parametrized self-organizing map (PSOM) is 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.

2 Decomposing the Inverse Kinematics Mapping

We will formulate the decomposition 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. As mentioned, our work relies on the assumption that the axes of the last three joints cross at a point, which holds for most 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 $\boldsymbol{\theta}$

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. If solutions are discrete, this can be handled by keeping track of the different solution branches. In the case of redundant robots, 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.

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,

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

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 three observations and the motion of the last two joints can be applied.

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 by means of the following two algorithms:

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

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

A similar strategy permits to perform on-line learning, i.e., learning that is integrated in normal working operation. This requires access to the inverse $\phi_0^{-1}(\nu)$, which gives the orientation that the argument ν produces when $\theta = \theta_0$. By using PSOMs [7], the learning of a function automatically makes available a proper estimation of its inverse and, therefore, a separate estimator for $\phi_0^{-1}(\nu)$ is not required.

4 Experimental results

We have used the PUMA robot as a testbed to validate our procedure in a controlled setting. The ranges allowed for the six joints [1] are as follows: $[-150, -10]$, $[-215, -100]$, $[-35, 80]$, $[-110, 170]$, $[-100, 100]$, $[-100, 100]$. Note that this defines a very large workspace.

Local Parametrized Self-Organizing Maps (PSOM) [7] with a subgrid size of 4 knots per axis have been chosen as an appropriate neural model to experiment with the decomposition approach.

For the control experiment (labeled "standard") we simply generate grids in the joint space covering the workspace above. Then we move the robot to the different configurations represented in the grid to obtain the associated positions and orientations. Thus, each knot in the grid requires one movement.

In the experiment to test our decomposition approach, we generate a grid for θ and move the first three robot joints to traverse each of its knots 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 ϕ_0 points.

Orientations and rotations are represented with five elements of the corresponding rotation matrix that determine it univocally.

Tables 1 and 2 show the precisions attained with an increasing number of movements. Units are millimetres for position and radians for orientation. 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 knot in the grids and to lower complexity in the optimizations performed in the PSOMs.

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.

In addition to learning efficiency, our decomposition procedure has other advantages over classic learning of IK in some contexts. For example, in [5] we tackled IK learning for a REIS robot placed in a Space Station mock-up, whose

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 1. Position (in milimeters) and orientation (in radians) precisions obtained with different numbers of movements using the standard procedure.

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 2. Position (in milimeters) and orientation (in radians) precisions obtained with different numbers of movements using the new decomposition procedure.

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 successfully completed.

References

1. Fu K.S., González R.C. and Lee C.S.G., 1987: *Robotics: Control, Sensing, Vision, and Intelligence*, New York: McGraw-Hill.
2. 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.
3. 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.
4. Ritter H., Martinetz T. and Schulten K.J. , 1992: *Neural Computation and Self-Organizing Maps*. New York: Addison Wesley.
5. Ruiz de Angulo V. and Torras C., 1997: Self-calibration of a space robot. *IEEE Trans. on Neural Networks*, **8(4)**: 951-963.
6. 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)**.
7. Walter J. and Ritter H., 1996: Rapid learning with parametrized self-organizing maps. *Neurocomputing*, **12**: 131-153.