EE572

Final Report

Distal Supervised Learning for Solving Inverse Kinematic Problems

Daniel Nikovski	-	*danieln@cs.siu.edu*
Sridhar Ramakrishna	-	*sridhar@zeus.c-engr2.siu.edu*
Rajani Kanth Koneru	-	*rajani@zeus.c-engr2.siu.edu*
Srinivas Jamhed	-	*srini@zeus.c-engr2.siu.edu*
Anil Konijeti	-	*anil@zeus.c-engr2.siu.edu*

Submitted May 12, 1994

Southern Illinois University at Carbondale

Abstract

The inverse kinematics of high degree of freedom (DoF) manipulators is one of the hardest problems in robotics. Neural networks seem to be a promising tool for its solution due to their ability to capture complex nonlinear mappings by learning example solutions and generalizing properly upon them over the whole problem space. However, a straightforward application of typical connectionist modeling techniques tends to yield poor solutions due to the nature of the mappings that have to be learned. Consequently, some major enhancements to the standard connectionist models are necessary in order to provide better solutions to the inverse kinematic problem.

The method of Distal Supervised Learning (DSL), proposed by Jordan and Rumelhart ([4]) is a variation of the well-known feed-forward architecture ([9]). DSL finds the inverse model of a controlled object by using its forward model. The results, reported in [4], however, are for the simple case of a 3 DoF planar arm. Our work investigated a much harder case of the inverse kinematic problem: modeling an industrial 6 DoF robot. It turned out that scaling up the problem did not require only bigger networks and larger training times, but also reconsidering the basic procedures, used to train the model. In particular, training the inverse model by applying DSL as described in [4] failed, while a simple change in the training method yielded a controller that successfully implemented the inverse kinematics of the robot being modeled.

In addition to that, we carried out a number of experiments to investigate the influence of the type of forward model on the attained accuracy of the controller. We demonstrate that some forward models lead to inaccurate controllers. Some ideas about how a proper forward model should be selected are discussed.

1 Statement of the problem

1.1 Inverse kinematic problem

The problem that we are concerned with is that of learning the forward and inverse kinematic of a robot arm ([1, 4]). As shown in Figure 1, the configuration of the arm is determined by the joint angles for each of its joints. This configuration thus determines the Cartesian coordinates x_1 and x_2 of the tip of the arm. The function that relates these variables is the *forward kinematic* function $\mathbf{x} = g(\theta)$. This function can be always computed unambiguously. Its practical calculation involves multiplying frame matrices [1].

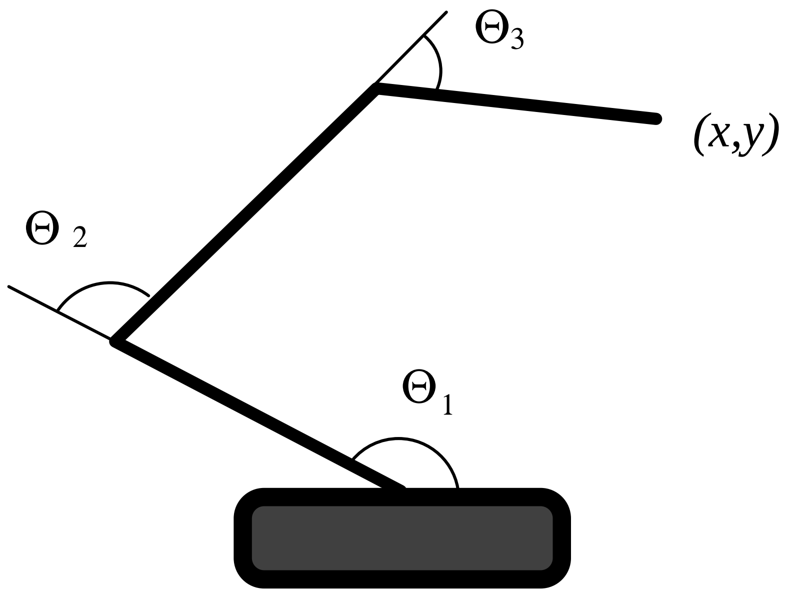


Figure 1: Inverse kinematic problem([1]).

The forward kinematic function $g(\theta)$ is a many-to-one mapping: that is, for every Cartesian position inside the boundary of the workspace, there are an infinite number of joint-angle configurations to achieve that position. In the general case the set of these configurations does not constitute a linear space, but a manifold of a general type. This means that the inverse kinematic relation $g^{-1}(\mathbf{x})$ is not a function; rather, there are an infinite number of inverse kinematic functions corresponding to particular choices of points θ in the inverse images of each of the Cartesian positions. The task of learning an inverse kinematic controller for the arm is that of finding a particular inverse among the many possible inverse mappings (any such solution will usually do unless we have additional constraints on the type of the solution).

1.2 Description of the test robot

The robot we used in the experiments was a 6 DoF anthropomorphic robot called OSCAR-6 ([10]). The coordinate frames of the robot are shown in Figure 2:

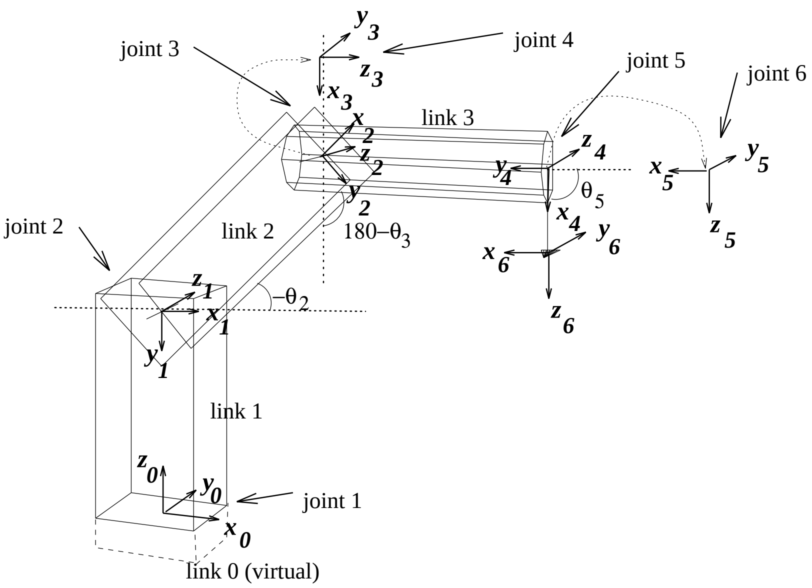


Figure 2: Line drawing representation of the OSCAR-6 robot used in the experiments ([10]).

In our experiments we used all six joints of the robot. Throughout our work we used a simulator of the robot, called Simderella, developed at the University of Amsterdam [10]. The simulator is implemented as a distributed system, communicating through Berkley sockets in UNIX environment. The simulator consists of a number of modules (Figure 3):

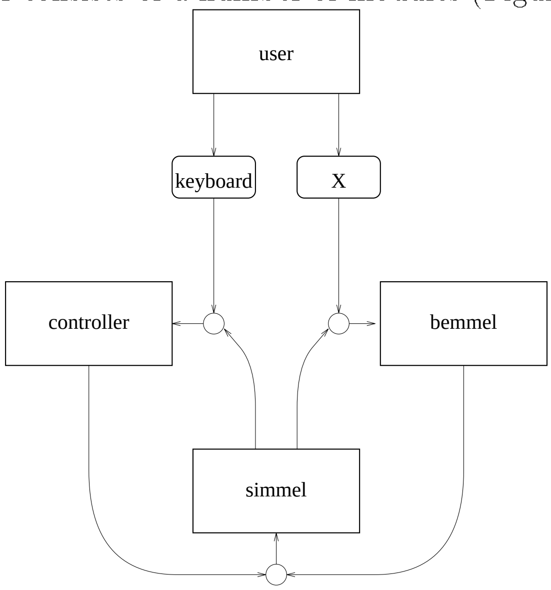


Figure 3: Program flow in Simderella ([10]).

For our problem we were interested only in the Simmel module, which simulates the forward kinematics of the robot by performing the necessary matrix computations ([1]). In particular, we have used a function that yields as its result the position in Cartesian space that corresponds to a given joint-angles configuration.

2 Literature review

A distinctive feature of the problem of solving the kinematic equations of a manipulator is that the problem is nonlinear. Given the numerical value of the coordinates of the tip of the manipulator it is necessary to find values of the joint angles [1]. As with any nonlinear set of equations, the primary concerns here are the existence of solutions, the multiplicity of the solutions and the method of solution.

The question of whether solutions exist or not is related to the concept of manipulator workspace. Workspace is that volume of space which the end-effector of the manipulator can reach. For a solution to exist, the specified goal point must be within the workspace. It is useful to consider two definitions of workspace ([1]): **dextrous workspace** is that volume of space that can be reached with all orientations. That is, at each point in the dextrous workspace, the end-effector can be arbitrarily oriented. The **reachable workspace** is that volume of space which the robot can reach in at least one orientaioin. Clearly, the dextrous space is a subset of the reachable workspace. Hence, a solution of the inverse kinematic problem is meaningful only if the goal is in the workspace of the robot.

Inverse kinematics is a hard problem in robotics in that no single best solution exists for all possible manipulators. As a result of this many methods of solutions have been proposed that can be classified roughly into three broad classes: **closed form solutions**, **numerical solutions** and **adaptive systems solutions**. These will be reviewed in the following subsections.

2.1 Closed form solutions

A closed form solution implies that we can find a solution method based on analytic expressions or on the solution of a polynomial of degree 4 or less, such that noniterative calculations suffice to arrive at a solutiuon. Within the class of closed form solutions we distinguish two methods of obtaining the solution: **algebraic** and **geometric**. These distinctions are somewhat vague, since geometric methods ultimately lead to algebraic expressions.

A sufficient condition that a manipulator with six revolute joints will have a closed form solution is that three neighboring joint axes intersect at a point ([1]). Almost every manipulator with six degrees of freedom built today has three axes intersecting, which allows for a closed form solution for this class of industrial manipulators. However, a general closed from solution is not generally available for manipulators with greater than six DoF.

2.2 Numerical solutions

Numerical methods are based on iterative approximation to the solution by optimization techniques. Because of their iterative nature, numerical solutions are generally much slower than the corresponding closed form solutions, which makes them practically unusable for real applications. Numerical solution to nonlinear equations is a whole field of study in itself, and is considered to be generally more powerful, even though slower than the analytical methods of solution. For example, it has been shown that all systems with revolute and prismatic joints having a total of six degrees of freedom in a single series chain are now solvable by numerical methods [1].

2.3 Neural networks for adaptive control

The solution methods based on neural networks form a class clearly distinct from the two classes of methods described above. Neural nets do not operate on analytical transformations; rather, their behavior is based on uniform nonlinear signal processing. Training neural networks does not involve analytical solutions to a set of equations - optimization in the weight space is the primary solution strategy.

The distinction between neural nets and numerical methods might not be so obvious, but certainly is an important one. While numerical solutions perform iterations at run-time and take a lot of time to arrive at a solution, neural networks, on the contrary, are very fast end efficient at run-time and might be even superior timewise to the analytical solutions due to their inherently parallel operation. Once a good neural network controller has been created, its operation is extremely efficient. Of course, the extensive training time of neural networks might be a practical problem, but this is another question.

The computational power of neural nets and their applicability to control problems has generated a significant amount of research recently ([8, 3, 5, 6, 11]).

2.4 Global regularization

One fruitful approach to the inverse kinematic problem has been the use of Kohonen self organizing feature maps for global regularization of the inverse function ([2]). DeMers and Kreutz-Delgado acknowledged the existence of two fundamentally different issues which result in the need for some form of regularization of the inverse kinematic problem - the existence of excess degrees of freedom (local ill-posedness) and the existence of multiple solution branches (global ill-posedness). For certain classes of manipulators, learning methods applied to input-output data generated from the forward function can be used to globally regularize the problem by partitioning the domain of the forward mapping into a finite set of regions over which the inverse problem is well posed. Local regularization can be accomplished by an appropriate parametrization of the redundancy consistently over each region. As a result, the ill-posed problem can be transformed into a finite set of well-posed problems. Each can then be solved separately to construct approximate direct inverse functions.

2.5 Direct inverse modeling and the nonconvexity problem

Direct inverse modeling treats the problem of learning an inverse model as a classical supervised learning problem. The idea is to observe the input/output behavior of the environment and to train the inverse model directly by reversing the roles of the inputs and outputs. Data are provided to the algorithm by sampling in action space and observing the results in sensation space [4].

However, a major problem arises when the input/output mapping behavior of the environment is many to one. In such a case the inverse mapping will map more than one images to a given point. A Least-Mean-Square classifier such as the backpropagation algorithm assumes that only one image exists for each point. In the case of a one-to-many mapping, it will associate the centroid of the multiple images with the input point. If the centroid does not belong to the manifold of the images (i.e., it is not a solution itself, the learning will result in large error (Figure 4).

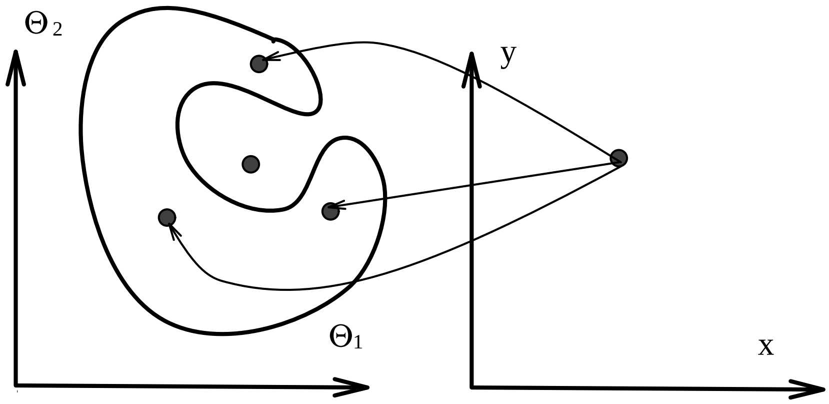


Figure 4: The nonconvexity problem([4]).

Another problem with direct inverse modeling is that the spatial density of the examples in the output space of the forward problem is not uniform. Since the output space of the forward problem is the input space of the inverse problem, this means that when using the inverse mapping the system will not generalize well in regions which have not been well represented during the training.

The two problems described above led Jordan and Rumelhart ([4]) to a new architecture for control that they called Distal Supervised Learning.

2.6 Distal supervised learning

Jordan and Rumelhart proposed an architecture of two cascaded neural networks, responsible for the inverse and forward models of the controlled plant (Figure 5).

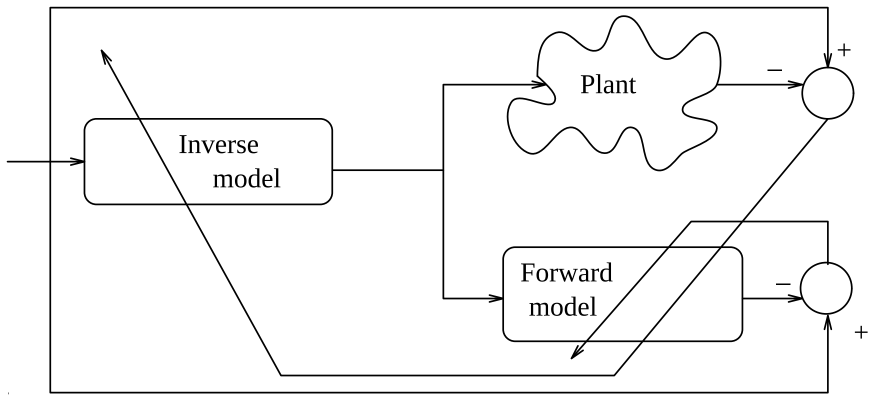


Figure 5: Distal supervised learning architecture.

The idea of Jordan and Rumelhart for solving the problem was to avoid the direct inverse modeling entirely. Instead, they proposed to train first a neural network to model the *forward* kinematics, after which to use this network to train indirectly the inverse model. They used the fact that the composition of the inverse and forward models must yield the identity function. So, if the weights of the forward model are kept fixed and only the weights of the inverse model are modified, the inverse model will be eventually learned if the training input-output pairs stand for the identity function. In this way, the effect is that only one of the possibly many solutions is chosen for a given target point.

3 Our approach and solution to the problem

For our work we have chosen the DSL method as a general architecture. Three distinct subtasks can be readily recognized - first, preparing training and testing sets, then training the forward model and finally training the inverse model.

3.1 Training and testing patterns

We quickly discovered that our attempt to model the inverse kinematics of a six DoF robot faced one substantial problem - the huge amount of training patterns that had to be learned by the forward model. The number of patterns with which we train a neural network is an important consideration when deciding about the number of hidden layers and the number of neurons in them. Generally, the number of adjustable weights grows coarsely linearly with the number of training patterns, and the two quantities are related through measures such as VC dimension or dichotomizing power. Hence, a large number of training patterns would probably require a large network. On the other hand, it could be seen easily that the number of training patterns grows very quickly with the number of sampling points per joint in this specific case of a six DoF manipulator. For instance, if we need only three angles for each joint, the number of training patterns will be $3^6 = 729$. If we increase them to 4 angles per joint, we get 4096, and further increasing the number of samples per joint leads to sets, which seem intractable by moderate size nets. The training time would grow also very quickly (experience has shown that it grows typically exponentially).

It should be noted that we need a forward model of substantial accuracy, because the forward model will be used to propagate back the error estimated by the real plant. We decided to sample the whole space uniformly, but with low spatial frequency - we took only three samples per joint angle. These include the minimum and maximum angle and one more angle that is the average of the minimum and the maximum ones. We chose the extremal angles, because neural nets seem to interpolate much better than they extrapolate from examples. This gave us 729 training patterns for the forward model.

As pointed out in previous sections, the forward kinematics is a many-to-one mapping. In our case, the 729 points in joint space map to only 112 points in Cartesian space. We have used these 112 points as input/output pairs for the two cascaded networks (respectively desired positions and obtained positions). For the testing set, another 584 points in Cartesian space were generated that sampled the *joint* space approximately uniformly. None of these points was present in the testing set.

3.2 Training of the forward model

The basic decisions we had to make were how many hidden layers and how many neurons in them were necessary, as well as what learning parameters were likely to get us to a good forward model. By trial and error we managed to train a forward model with two hidden layers, ten neurons in each. We used simple steepest descent optimization with the hope that it is robust enough to arrive at an acceptable solution. It was desirable to have a faster optimization method such as Conjugate Gradient, and to this end we tried to install two public domain simulators - Xerion and the Stuttgart Neural Network Simulator. Unfortunately, we could not install them due to technical difficulties with X Windows. Even with simple gradient descent, we achieved a mean squared error (MSE) of 0.000507, which we consider to be a very precise forward model. The learning rate was set initially to 0.6 with a momentum of 0.7, which led us to a local minimum. The learning rate was then increased to 1.2 and after escaping the local minimum reduced to 0.05 with a momentum of 0.9 in order to fine tune the solution.

3.3 Training of the inverse model

Training the inverse model was much harder than that of the forward model. In the first place, we never managed to get an acceptable MSE by directly using DSL. The simulator kept getting stuck in a local minimum corresponding to a MSE of about 0.02 (clearly unacceptable as a solution). It seemed that all random points in weight space around the origin (small initial weights) tended to converge to this local minimum.

We tried an alternative approach then. We trained the inverse model with the *predicted perfor-*

mance error rather than with the *performance error*. That is, we ignored the plant whatsoever initially and found a good inverse of the forward model - the MSE went down to 0.000992. This inverse, of course, was not what we needed ultimately - the predicted performance error is an unreliable estimate of the performance of the controller ([4]). In accord with this observation, our inverse model, tested on the real plant, gave an MSE of 0.01652, which was much worse a result. What was of importance, however, was the fact that this MSE was below the initial local minimum. The next thing to do was simply to start with this imprecise inverse model and finely tune it on the real plant. We managed to lower the MSE to a value of 0.00661, which is significantly better than the starting inverse controller, but is still higher than what we got for the forward model. This value was low enough to conclude that we had learned the examples from the training set. A simple statistical procedure supports this statement: we estimated experimentally the MSE that can be obtained by random weights and the standard deviation of this MSE. Our result for MSE was way below three standard deviations from the estimated value for MSE. Since we had a number of networks that had this property, we could safely conclude that getting such low values for MSE is not due to chance, but to learning the training examples.

The neural net that we used had three hidden layers with two, three and three neurons respectively in the hidden layers. The learning rates were small (in the range 0.01 to 0.1), while the momentum was kept around 0.8-0.9.

3.4 Testing on independent targets

The test on independent targets gave MSE values comparable to those for the training patterns. The 584 new points gave an MSE of 0.0116, which is still below three standard deviations from the expected value for MSE on a random network (the expected MSE and the standard deviation were estimated to 0.0252 and 0.0046 respectively, from ten random nets).

Since the size of the testing set is sufficiently large, we can conclude that the controller successfully generalizes over the training patterns and presents a successful solution to the inverse kinematic problem for this six DoF robot.

4 Some further experiments with DSL

As we saw, DSL is a powerful method that solves successfully the inverse kinematics of complex robots. However, we expect that the quality of the forward model plays an important part in the learning process. We can conjecture that certain types of forward models will actually *prevent* DSL from learning the inverse of a plant. Let us illustrate this with a set of experiments, exploring a simple case of the inverse kinematic problem.

Experimental testbed. We chose the simplest possible configuration of a robot arm - that of a planar arm with two rotational joints and links of length 0.5 each. Thus the operational space of the robot is a circle of unit radius. Even this simplest manipulator is redundant - for each target point in the operational space two configurations of the joint angles exist that result in the the tip of the arm at this point (e.g. point A in Figure 6).

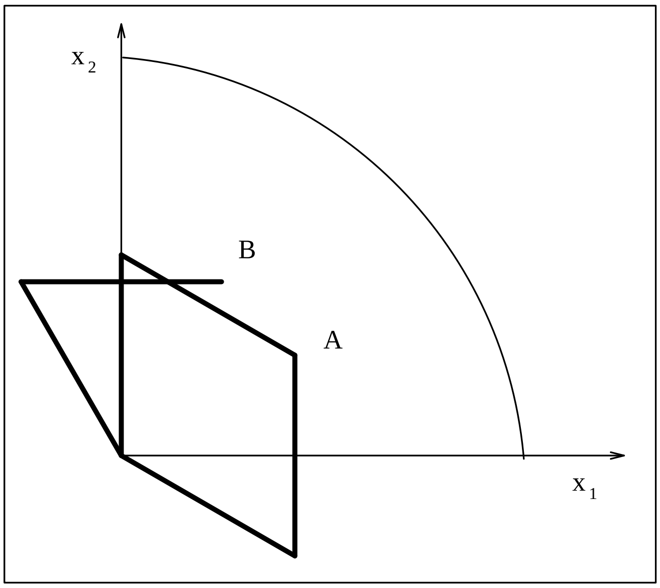


Figure 6 Training vectors. Two solutions for point A are shown

As Jordan and Rumelhart pointed out ([4]), it is *impossible* for the Back Propagation algorithm to learn the inverse kinematics dependency in this case, for a very simple reason - the inverse kinematics is merely not a function. Indeed, for point A, two distinct input/output pairs can be constructed that correspond to the two possible mappings from joint angles to point A:

$$\theta_{A_1} \Rightarrow \mathbf{x}_A$$

$$\theta_{A_2} \Rightarrow \mathbf{x}_A$$

$$\theta_{A_1} = (-\pi/6, 2\pi/3)$$

$$\theta_{A_2} = (\pi/2, -2\pi/3)$$

$$\mathbf{x}_A = (\frac{\sqrt{3}}{4}, \frac{1}{4})$$

The forward kinematics can be learned easily by giving as input the joint vectors $\theta_{1,2}$ and as output the Cartesian vector $\mathbf{x}$ that gives the position of point A. However, if we try to learn the inverse kinematics by swapping θ and $\mathbf{x}$ as output and input, this will not work, because in this case we will be trying to match the vector $\mathbf{x}$ to *two* different output vectors at the same time. The mean squared error (MSE) will never get to zero, no matter what the weights of the approximating neural network are [4].

DSL is a powerful technique for modeling inverses of functions that are not one-to-one. It provides a way to reduce the MSE to an arbitrary small value, thus assuring *repeatability*, in robotics terms, of the kinematics being modeled, i.e. all training positions will be perfectly learned and repeated whenever this is necessary [1]. However, this is not the ultimate goal of the whole enterprise. We actually need not only repeatability, but most important, we need *accuracy*, i.e good generalization on unknown input. We can show that DSL *does not* always provide accuracy by means of a simple example .

We have trained two separate networks to model the forward kinematics of the arm in Figure 6. The region of interest are points A and B in Figure 6 and the straight line between these points. As we noted, two possible solutions are possible for each target point. Let us fix a solution for point B (the one shown in Figure 6), and train the first net with the input-output pairs (Figure 7a)

$$\theta_{A_1} \Rightarrow \mathbf{x}_A$$

$$\theta_B \Rightarrow \mathbf{x}_B$$

$$\theta_{A_1} = (-\pi/6, 2\pi/3)$$

$$\theta_B = (2\pi/3, -2\pi/3)$$

Note that in this case, the solutions θ_{A_1} and θ_B are "dissimilar", i.e. they are not of the same type. The second forward model uses the other solution for point A (Figure 8a):

$$\theta_{A_2} \Rightarrow \mathbf{x}_A$$

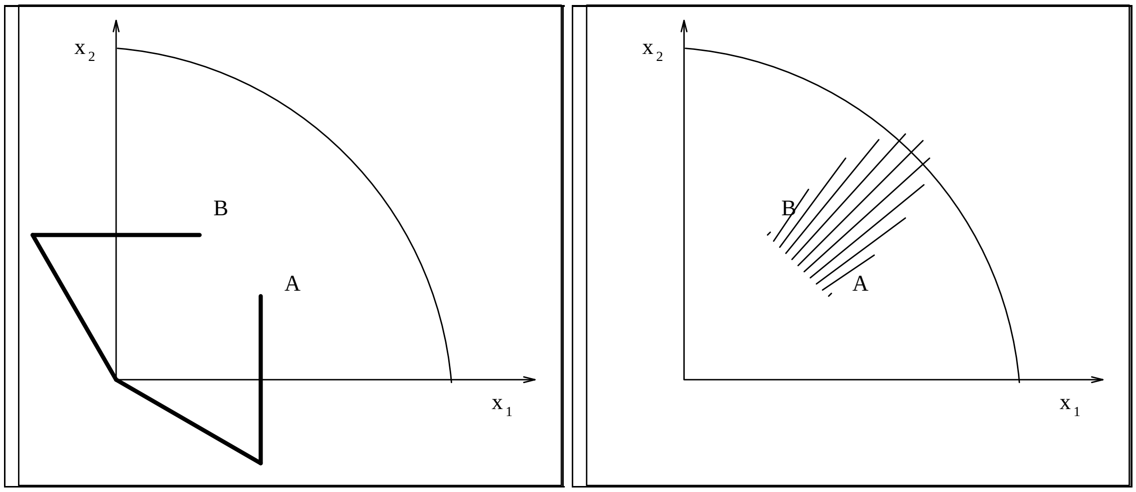
$$\theta_B \Rightarrow \mathbf{x}_B$$

$$\theta_{A_2} = (\pi/2, -2\pi/3)$$

$$\theta_B = (2\pi/3, -2\pi/3)$$

Here the solutions are similar (the second angle has the same value).

Having trained the two separate forward models, we used them to train two inverse models by DSL. As test vectors, we used nine points on the straight line between points A and B (together with A and B for experiment control). If an inverse model is good, we would expect that it will generalize well on these points, which are a linear combination of the two vectors A and B used for training. The results are presented in Figures 7b and 8b:



a) b)

Figure 7. a) Forward model with training vectors θ_{A_1} and θ_B b) Error field of DSL, using a) as forward model

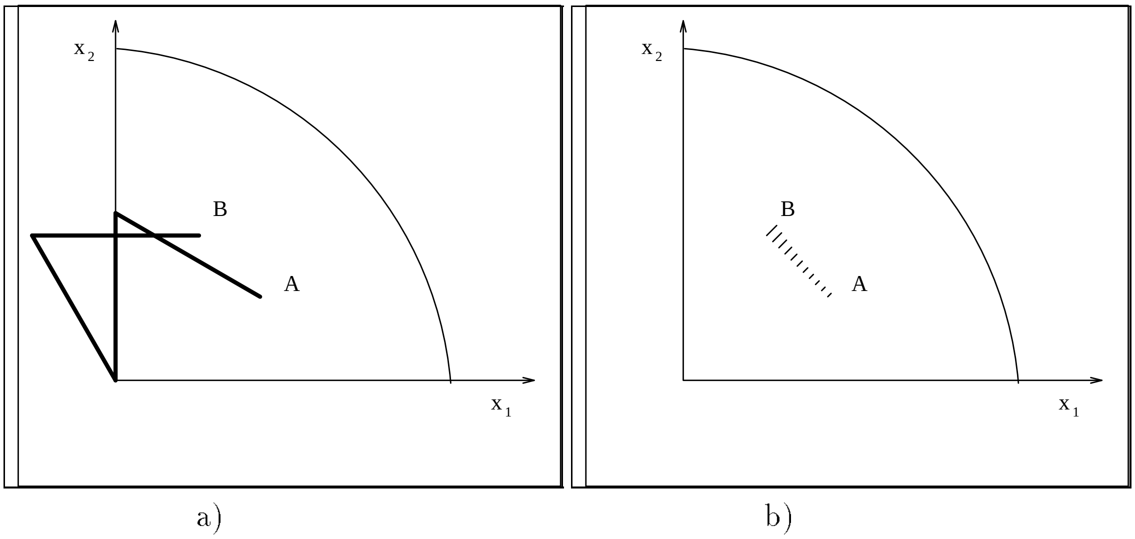


Figure 8. a) Forward model with training vectors θ_{A_2} and θ_B b) Error field of DSL, using a) as forward model

It can be seen that the choice of forward model in DSL has an enormous effect on the generalizing capabilities of the resulting network. We can hardly speak of smooth generalization in the first case, when two different types of solutions were used in the forward model (Figure 9). The arm changes its configuration, going through a zero angle for the second joint (straight arm), which results in a very large error. On the other hand, in the second case DSL the error of approximation is almost the same for all points on the test line.

We believe that the difference between the two forward models is the key factor in the generalizing abilities of networks trained with DSL. A proper choice of training vectors for the forward model results in good generalization, while a choice that involves more than one type of inverse solution results in bad generalization at the boundary between the points by which the solutions have been specified.

The same effect is observed when we do straightforward inverse modeling, as in [7]. We performed direct inverse modeling for the two cases (Figure 7a and Figure 8a), and the results look similar to those from Figures 7b and 8b.

5 Conclusions

The method of distal supervised learning proved to be successful in learning the inverse kinematics of a six DoF anthropomorphic robot. Even though the joint space was undersampled in our experiments, the training patterns were enough for the controller to capture the nature of the inverse transformation and generalize throughout the joint space. Of course, an industrial controller needs better accuracy than the one that we achieved and further training can positively improve the accuracy of the learner, especially if optimization methods such as Conjugate

Gradient are used. Our goal in this project was to test if DSL can be applied to the inverse kinematics of realistic robots and we think that our experiments give a convincingly positive answer.

We found that increasing the number of hidden layers yielded better results than increasing the neurons in them. One possible explanation of this lies in the fact that what we are trying to learn is not a classification problem with many decision regions, but a continuous, nonlinear mapping.

Having identified the problem with the choice of forward models, we can try to provide in our future work a rigorous analysis of its nature (e.g. prove that a *lower* bound of the generalization error depends on the training set for the forward model, or in other words, prove that if we have a bad forward model, no matter what we do, the generalization will be poor.) Another, more ambitious objective will be to provide a method for choosing the forward model such that the generalization error can be guaranteed to be in certain arbitrary small bounds.

For the time being we can suggest that the forward model should incorporate only one type of solution to avoid inaccuracy of the controller. For the experiment above, if the second angle is restricted to be only positive, the problem will become well-defined. It is not clear, however, how this can be extended to the general case with arbitrary number of DoF.

References

[1] J.J. Craig. *Introduction to Robotics*. Addison-Wesley, Reading,MA, 1986.

[2] David DeMers and Kenneth Kreutz-Delgado. Learning global direct inverse kinematics. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 589–594. Morgan Kaufmann, 1992.

[3] T. Fukuda, T. Shibata, M. Tokita, and T. Mitasuoka. Adaptation and learning for robotic manipulators by neural networks. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 3283 – 3288, 1990.

[4] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307 – 354, 1992.

[5] A. Kuan and B. Bavarian. Using adaline neural networks for compensation of structured uncertainty in manipulators. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 271 – 278, 1992.

[6] S. Y. Kung and J. N. Hwang. Neural network architectures for robotic applications. *IEEE Transactions on Robotics and Automation*, 5(5), 1989.

[7] M. Kuperstein. Neural model of adaptive hand-eye coordination for single postures. *Science*, 239:1308–1311, 1988.

[8] W.Thomas Miller, Richard S. Sutton, and Paul J. Werbos. *Neural networks for control.* MIT Press, Cambridge, MA, 1990.

[9] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel distributed Processing: Explorations in the microstructure of cognition*, volume 1, chapter 8. MIT Press, 1986.

[10] Patrick van der Smagt, Frans Groen, and Ben Kröse. Robot hand-eye coordination using neural networks. Technical Report TR CS-93-10, University of Amsterdam, 1993.

[11] A. Y. Zomaya, M. E. Suddaby, and A. S. Morris. Direct neuro-adaptive control of robot manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, 1992.