Proceedings of the 1991 IEEE
International Conference on Robotics and Automation
Sacramento, California - April 1991

Neural Network Learning of the Inverse Kinematic Relationships for a Robot Arm

Stuart Kieffer, Vassilios Morellas and Max Donath

Robotics Laboratory, Dept. of Mechanical Engineering / Productivity Center
University of Minnesota, Minneapolis, MN 55455

Abstract

A methodology is presented whereby a neural network is used to learn the inverse kinematic relationship for a robot arm. A two link, two degree of freedom planar robot arm is simulated, and an accompanying neural network which solves the inverse kinematic problem is presented. The method is based on Kohonen's self-organizing mapping algorithm using a Widrow-Hoff type error correction rule as introduced by Ritter, Martinetz, and Schulten. In this paper, we have specifically addressed a number of issues associated with the inverse kinematic solution, including the occurrence of singularities and multiple solutions. Simulation results for a planar two degree of freedom arm provide evidence that this approach is indeed successful. The approach is a significant improvement over other neural net approaches documented in the literature.

Introduction

The solution of the inverse kinematic problem maps the six-degree of freedom world coordinates of the robot manipulator's end-effector into the robot's joint space. The number of solutions will depend on the manipulator configuration (including the number, type, and relative location of the joints), on the range of motion of each of the joints, and on the location of the selected end-effector position in world coordinates. There may be no solutions, a unique solution, or multiple solutions. Kinematicians have for some time worked on this problem and have, as of yet, not developed a methodology that can solve the inverse kinematics problem for a generalized N-degree-of-motion freedom manipulator [3]. Solutions have been found, however, for certain manipulator configurations. Industrial robot manipulators have conformed to these configurations in order to facilitate the specification of tasks in a Cartesian-defined workspace (often called the task space).

An alternative solution to that of developing and solving a set of equations would be useful for those cases where:

a) the equations cannot be derived even though the manipulator can be designed and built,
b) the equations are so computationally intensive that solutions take too long to compute for practical robot control implementation, and
c) the equations involve coefficients which cannot readily be determined.

Humans and animals control their extremities without recourse to solving a set of equations. There ought to be a methodology that more closely mimics the mechanisms whereby we move our arms without consciously determining (i.e., calculating) the necessary joint angles, and velocities that enable that motion. An acceptable solution must recognize the existence and location of singularities and find a valid solution such that the continuity of the mapping between world and joint space is preserved. Possible solutions to this problem include both numerical procedures and neural network based methods. The success of numerical solution procedures depends to a great extent on the formulation of a mathematical expression that accurately describes the functional relationship between the input parameters (specified end-point position of the manipulator in world coordinates, in our case) and the output solution parameters (the joint angles in our case). There are similarities between traditional numerical solution procedures and neural net methods. These include the existence of an iterative adaptation procedure and a performance measure. However, we wish to limit ourselves to neural net procedures in which the solution is not determined based on a mathematical expression defining the input/output relationship, but is captured in some form of an associative memory relationship.

Several neural network approaches have been proposed in the literature. Guez and Ahmad [4], applying the back-error propagation algorithm based on a three layer perceptron, solved the problem as a learning process. Their first approach "yielded good results but were [*sic*] not accurate enough to be practically utilized." A second attempt by these authors combined back-error propagation with a conventional numerical procedure. They used the neural network simply as a "lookup table in providing a good initial guess to an iterative procedure." [1]. In

general, it should be noted that back propagation requires the development of "hidden units," which will slow down the learning process. Guo and Cherkassky [5] proposed a solution using a Hopfield net. Their solution did not directly develop the inverse kinematic relationship, but instead coupled the neural net with a Jacobian based control technique. The approach that comes closest to the one used in this paper, at least conceptually, is that proposed by Miller [8]. He developed an adaptive system for learning control, based on the CMAC (Cerebellar Model Articulation Controller) model, first introduced by Albus [2]. CMAC is a perceptron based classifier that uses a two stage mapping from the input space into the output space. The two stages are needed because of the enormous amount of memory that would otherwise be necessary. By doing this, a many-to-one mapping is used to reduce the needed memory. This limits the performance of the CMAC module. At the time that CMAC was developed, learning by back-propagation had not as yet been developed. Miller used the CMAC module together with a Widrow-Hoff like rule for adaptation. One disadvantage of the CMAC method is that its performance must always be evaluated experimentally. Kuperstein and Wang' s work [7] also has similarities to the method proposed by Ritter et al. [9], [10]. They developed a neural controller that learns to move and position a link carrying an unforseen load. They use a mapping from the input space of neural elements, (associated with values that are functions of the initial and final position of the end-point and a measurement of the payload), into the output space of actuator torque signals through an array of modifiable weights. Their approach does not deal with the kinematic issues of path generation but focuses on dynamic compensation alone.

The method used in this paper, Kohonen's self-organizing mapping algorithm [6] which uses a Widrow-Hoff type error correction rule, was first shown to be a valid approach to solving the inverse kinematic problem by Ritter et al. [9], [10]. This unsupervised method learns the functional relationship between input (Cartesian) space and output (joint) space based on a localized adaptation of the mapping, by using the manipulator itself under joint control and adapting the solution based on a comparison between the resulting location of the manipulator's end-effector in Cartesian space with the desired location. Even when a manipulator is not available, the approach is still valid if the forward kinematic equations are used as a model of the manipulator. The forward kinematic equations always have a unique solution, and the resulting neural net can be used as a starting point for further refinement when the manipulator does become available. The technique is independent of arm configuration, including the number of degrees of freedom and the link geometry. Interestingly enough, the method also "learns" the Jacobian matrix.

We will show that the method can be used for different workspace regions by dynamically learning the functional relationship between the specified Cartesian coordinate workspace and the manipulator joint space. In this study, we will investigate the problems associated with the occurrence of multiple solutions and singularities. Can the system "learn" that multiple solutions may exist? Can the system "learn" that it cannot move an arm beyond its reach? We will also examine the performance of the algorithm. Simulation results that further validate the approach and a discussion of the continuity of the mapping will be presented together with comments on accuracy and the speed of convergence.

Description of The Model

As a test bed for our experiments we used a two-degree of freedom, planar robotic arm with links l_1 and l_2 as shown in Fig. 1. The joint angles are denoted by q_1 and q_2 and the Cartesian coordinates of the end-point by x and y. Fig. 1 shows the workspace that can be reached by the end-effector (in gray) assuming no joint limits.

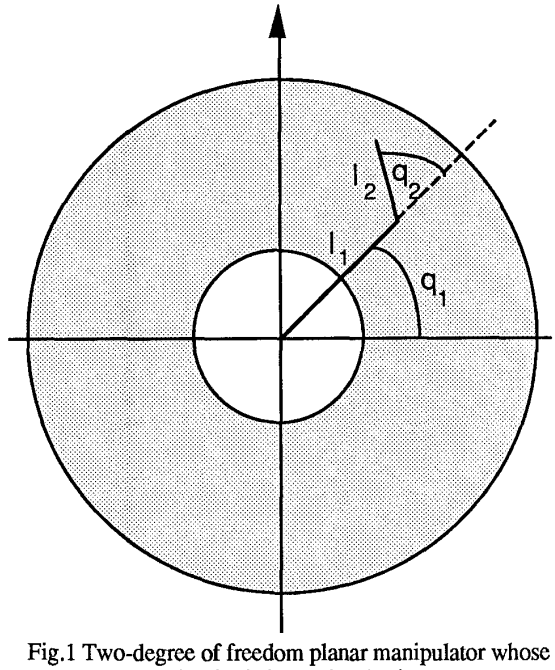

Fig.1 Two-degree of freedom planar manipulator whose end-point is located at (x,y)

For our experiments, the forward kinematic equations (1) and (2) replace Ritter's et al. [9],[10] "vision system." These equations can be considered as a "hidden" supervisor. For a human extremity under neuromuscular control, the equations represent the perception of where the hand is for a conscious change in each of the joints. For a manipulator, these equations model the measured displacement of the end-effector's location in Cartesian space which results from a commanded displacement of each joint.

$$x = l_2 \cos(q_1 + q_2) + l_1 \cos(q_1) \quad (1)$$
$$y = l_2 \sin(q_1 + q_2) + l_1 \sin(q_1) \quad (2)$$

Although this case is a simple one, it nevertheless becomes a valuable illustrative experiment which captures the essential problems associated with far more complex manipulator configurations. Attempting more complex configurations makes no sense if we cannot develop an understanding of the approach's power and limitations for readily understandable configurations.

The Algorithm

The algorithm is based on an "extension of Kohonen's self-organizing mapping algorithm, which is capable of learning a mapping between a (sensory) input space and a (motor) output space by establishing a topology-conserving map on a (usually) planar array of neuronal units" [9]. A topological feature map is a data structure where the"interrelationships of the data are captured in the spatial arrangement of the corresponding neurons"[9]. The map defines the ordering of neurons, allowing the algorithm to freely develop within this structure.

The description that follows describes the situation for a two jointed manipulator whose end-point is specified by only two degrees of freedom in Cartesian space, x and y. In the general case, any number of joints may exist and up to six degrees of freedom can be specified for an end-effector's position and orientation. Each neuron **r**, distinguished by its indices (i,j), is a unit of a general network (planar in our case) that has associated with it a 2-dimensional vector $\mathbf{w_r}$ = (x,y), where the values x, y represent the neuron's physical location within the Cartesian input space. Initially each neuron **r** = (i,j) is assigned a random position, $\mathbf{w_r}$, uniformly distributed within this input space. Kohonen's self-organizing topology-conserving mapping algorithm is the procedure that orders the neurons so that they conform to their spatial locations. The algorithm is extended by associating every neuron with another mapping data structure $\mathbf{u_r} = (\mathbf{Q_r}, \mathbf{A_r})$, where $\mathbf{Q_r} = (q_{r1}, q_{r2})$ is a vector in the joint space, and $\mathbf{A_r}$ is a matrix containing the change in Cartesian location resulting from a small change in joint coordinates. The size of $\mathbf{A_r}$ is equal to the product of the specified degrees of freedom in Cartesian space and the number of joints. In our case, $\mathbf{A_r}$ has four elements. The elements of $\mathbf{A_r}$ are, in effect, the elements of the Jacobian matrix which relates the input $\mathbf{w_r}$ to the output $\mathbf{Q_r}$. Initially, the values assigned to the data structures' variables $\mathbf{w_r}$ and $\mathbf{u_r}$, are uniformly random. The learning procedure will modify (i.e. learn) the values of the variables $\mathbf{w_r}$, $\mathbf{u_r}$, $\mathbf{A_r}$ so that the relationship described by equation (3) is finally satisfied.

$$\mathbf{Q_r} = \mathbf{A_r}\,\mathbf{w_r} \quad (3)$$

The Learning Procedure

The exact space (Cartesian/input space) that the robotic arm is to learn must first be defined. We impose on this space a rectangular grid (in this planar case) or **W**-map of neuronal units. The dimensions of the **W**-map determine the number of neurons. As the number of neurons is increased , a better resolution of the map is achieved. Each neuron, **r**, is associated with a vector $\mathbf{w_r}$=(x,y) that corresponds to a position within the input space as described earlier. The values of these vectors are initially assigned randomly. Each neuron,**r** , of the same grid is also associated with a record, $\mathbf{u_r} = (\mathbf{Q_r}, \mathbf{A_r})$, which forms the **U**-map, of joint angles, $\mathbf{Q_r} = (q_{r1}, q_{r2})$, and $\mathbf{A_r}$, the terms of the Jacobian corresponding to the node under consideration. These are all filled with random initial values. These two steps establish the framework within which we can work out our learning algorithm. The ultimate goal is the mapping of the input space (**W**-map) for each neuron into its output space (**U**-map) so that **w** and **u** are related through the functional relationship that is to be determined (the solution of the inverse kinematics problem (3)).

The learning process starts by presenting sequential inputs, which are uniformly random within the specified workspace region under study, to the **W**-map. The neuron, **r** which is "closest" to the presented input is then selected. "Closest" is defined by the minimum Euclidean distance between the input value **x** and the vector $\mathbf{w_r}$. The **w**-vector corresponding to the selected neuron, **s**, as well as those of the neurons that belong to its "topological neighborhood" will be updated as described below. The "topological neighborhood" $\mathbf{N_S}$ (Fig. 2) determines the neurons that together with the selected neuron will have their **w**-vector values updated every time a new input is presented. This neighborhood is a function of time, $\mathbf{N_S} = \mathbf{N_S}(t)$, and its shape is a square. Other shapes (rectangle, hexagonal, etc.) are also possible.

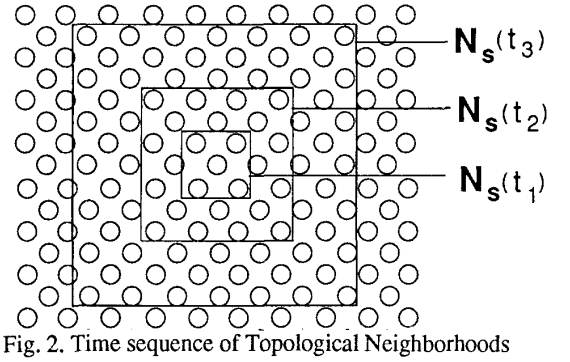

Fig. 2. Time sequence of Topological Neighborhoods $t_1 > t_2 > t_3$

Time is measured in learning steps. A step (cycle) occurs when an input is presented and the selected neuron and its neighbors are updated according to equations (4) and (7).

The following equation (4) determines the adaptation of the **w**-vectors.

$$\mathbf{w}_r^{new} = \mathbf{w}_r^{old} + \varepsilon\, h_{rs}\,(\mathbf{x} - \mathbf{w}_r^{old}) \quad (4)$$

where

x is the presented input vector, x=(x,y)

ε is a measure of the learning step size,

and

h_{rs} is a Gaussian distribution centered around the selected neuron s.

The parameter h_{rs} is given by the Gaussian distribution equation (5).

$$h_{rs} = \exp(-\|\mathbf{w}_r - \mathbf{w}_s\|^2 / 2\sigma^2(t)) \quad (5)$$

where the standard deviation of the Gaussian, $\sigma(t)$ is expressed as a function of time (learning step number),

$$\sigma(t) = \sigma_i(\sigma_f/\sigma_i)^{t/t_{max}} \quad (6)$$

where σ_i and σ_f are the initial and final standard deviation values of the Gaussian respectively, t is the number of learning steps already performed at that moment, and t_{max} is the total number of learning steps to be performed.

The variable ε in equation (4) is described similarly to equation (6), where

$$\varepsilon(t) = \varepsilon_i(\varepsilon_f/\varepsilon_i)^{t/t_{max}} \quad (6a)$$

where ε_i and ε_f are the initial and final learning step sizes. The range of both ε and σ is between 0 and 1.

The U-values, $\mathbf{u}_r = (\mathbf{Q}_r, \mathbf{A}_r)$, that correspond to the selected neuron and its neighbors are also updated according to equation (7).

$$\mathbf{u}_r^{new} = \mathbf{u}_r^{old} + \varepsilon' h'_{rs}\,(\mathbf{u}^* - \mathbf{u}_r^{old}) \quad (7)$$

where ε' and h'_{rs} are given by expressions similar to those of ε and h_{rs} respectively. The values of the parameters used in the experiments were $(\varepsilon'_i,\varepsilon'_f) = (\varepsilon_i,\varepsilon_f) = (0.5,0.0025)$ and $(\sigma'_i,\sigma'_f) = (\sigma_i,\sigma_f) = (0.5,0.0025)$.

u* is an improved estimate calculated as follows:

The joint angles $\mathbf{Q}_s$ associated with the selected neuron s are used in the forward kinematic equations (or using the actual manipulator under joint control) to obtain a "gross movement", which moves the end-effector to the position $\mathbf{x}_l$. This is followed by a "fine movement" again using the forward kinematic equations, with $\mathbf{Q}_l = \mathbf{Q}_s + \mathbf{A}_s(\mathbf{x} - \mathbf{w}_s)$, (or using the manipulator) which position the end-effector at a new position $\mathbf{x}_f$. The improved estimates for the joint angles and the Jacobian which form **u*** are then calculated using a *Widrow-Hoff* type correction rule, as given by equations (8) and (9) respectively.

$$\mathbf{Q}^* = \mathbf{Q}_s + \mathbf{A}_s(\mathbf{x} - \mathbf{x}_f) \quad (8)$$

$$\mathbf{A}^* = \mathbf{A}_s + \mathbf{A}_s(\mathbf{x} - \mathbf{w}_s - \mathbf{x}_f + \mathbf{x}_l)(\mathbf{x}_f - \mathbf{x}_l)^T \|\mathbf{x}_f - \mathbf{x}_l\|^{-2} \quad (9)$$

Simulation Results

The robot model (Fig. 1) that we investigated was configured with links of magnitude l_1=1.0 units and l_2=0.5 units. The maximum number of learning cycles was selected to be t_{max} = 20,000. This parameter controls the trade off between the time needed for convergence and the accuracy of the solution. A convergence test could have readily been designed for the case when all the specified Cartesian positions are within the workspace of the manipulator. However, the most interesting aspect of our investigation was to determine whether the learning paradigm described here can indeed recognize that a specified input end-point location outside the range is not within reach. In such cases, there will be a "residual" error between where the robot actually moves its end-point associated with the U-map (probably at its maximum reach) and the location where one has asked it to move (associated with the neuron's W-map). This residual error will confuse a convergence test based on monitoring the relative error. The tests were run for cases in which the presented inputs were chosen to be uniformly random in a square area.

For each test two graphs are plotted illustrating the performance of the system. When interpreting the x-y map of neurons in the first graph, the black circles which represent the **w**-vector are an indication of where you might wish the robot to go. The squares represent the locations that the robot moves to based on the learned **u**-vector. The second graph of q_1 vs q_2 plots the two elements of $\mathbf{Q}_r$, part of the learned $\mathbf{U}_r$ vector. The purpose of this plot is to demonstrate the non-linear continuity of the inverse kinematic mapping (a conformal mapping from $\mathbf{W}_r$ to $\mathbf{Q}_r$).

Three different situations were tested. Fig. 3 illustrates the locations of the four cases. In Case 1 the specified inputs fall totally within the robot's reachable space. Cases 2 and 3 illustrate the part of the workspace that was investigated in order to determine how the system behaves at either the outer or the inner reach of the end-point. Figures 4a and 4b show the performance of the system for Case 1 with a 12x12 grid of neurons after 12,000 cycles. The mean value for the errors along the x and y directions were found to be 0.0044 and 0.0043 respectively. The standard deviation for the errors was 0.0030 and 0.0033 along the x and y axes respectively. These errors can be reduced further by allowing additional

learning. Note that there are clearly edge effects, in which the errors are larger at the boundary of the test region than at the center.

In the second case, the inputs fell partially in the manipulator's reachable workspace, and partially outside of the outer boundary within the first quadrant. The learned system response for this case (Fig. 5) shows that the system was able to determine the inverse kinematic solution when the inputs were within reach. The theta1 vs theta2 graph in this case is shown to be a regular curvilinear grid for the points that are reachable. There is no inverse kinematic solution for points outside the reach of the manipulator, nor does a Jacobian exist. As a result, the U-vector for humans associated with $\mathbf{W_r}$ vectors outside the reach will respond with arbitrary locations in the field. A simple test can be instituted to check the Cartesian location resulting from the $\mathbf{u_r}$-vector against $\mathbf{w_r}$. If the error is greater than some set value, the response would be that no solution exists.

In the third case, the inputs fell partially within the manipulator's reachable space, and partially outside the inner boundary of the first quadrant. In this case only an 8x8 grid of neurons was used. Similar conclusions to that of the second case can be derived from Fig. 6a and 6b. Despite the fact that there were fewer neurons covering approximately the same area as in Case 2 of Fig. 5, the results indicate that the $\mathbf{u_r}$ vector mapped the unreachable $\mathbf{w_r}$ nodes directly on to the inner boundary of the workspace. As expected, the edge effects are more significant when fewer nodes are used.

The ability of the system to find multiple solutions was also verified. Figures 7a and 7b illustrate that the system can learn two possible sets of solutions. The right hand and left hand solutions were found by only changing the starting values assigned to the variables of the system. Although more cycles could have been run to improve the accuracy, we terminated the runs once it was clear that the two different solutions of the inverse kinematic relationship had emerged.

Conclusions

This paper evaluates the Ritter et al. approach to learning the inverse kinematic solution under conditions where the solution is not well behaved. A non-redundant two degree of freedom planar robot arm model was used. The results showed that this method is indeed successful. The large number of learning steps is not a problem since these can first be run using an off-line simulation in order to learn the values based on the model. The results can then be used as starting values in the second phase where an actual robot manipulator is used in order to experimentally determine the changes in the neurons due to unmeasurable or unmodellable characteristics. Finally once thes neurons are ordered based on some tolerance specifications, no further learning steps are required and the neurons can be used directly on-line to continuously transform the specified Cartesian locations to joint coordinates.

The resolution is only a function of the number of neurons used. The accuracy obtained was found to be a function of the number of learning steps. The use of a topological neighborhood about the selected neuron that decreases in size with increasing learning steps was necessary in order for the ordering of neurons to be correct. Continuity of the mapping was verified for the cases where the inputs were totally in the robot's reachable space. Multiple solutions to the inverse kinematics were correctly identified by simply changing the initial-random values for the $\mathbf{w_r}$ and $\mathbf{u_r}$ vectors. Although we did not examine the accuracy of the Jacobians that were learned, the fact that the system was able to correctly learn $\mathbf{Q_r}$ based on $\mathbf{A_r}$, implies that the Jacobian estimates, $\mathbf{A_r}$, were correct. A more detailed examination of the accuracy of the estimated Jacobian is in order.

When a training region was selected to cover a region which straddled the inner or outer boundaries of the reachable workspace, the system was still able to correctly learn the $\mathbf{w_r}$ and $\mathbf{u_r}$ vectors for all the neurons which were inside the workspace. For positions specified outside the workspace, no solution to the inverse kinematics exists. As such, no adaptation mechanism can work for these cases. What is most important is that a simple test can identify and flag these cases. There are still other issues to be addressed including the investigation of this approach for computing the full six-degree of freedom situation (both for the case where a closed form solution exists and for the case where it does not) and for dealing with redundant manipulator configurations.

References

[1] Ahmad Z. and Guez A., 1990. "On the Solution to the Inverse Kinematic Problem", IEEE Int'l Conf. on Robotics and Automation, Cincinnati, Ohio, Volume III pp. 1692-1697, May 1990.

[2] Albus J. S., 1975. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", Trans. ASME, J. Dynamic Systems Measurement and Control, vol 97, pp. 220-227 Sept. 1975.

[3] Duffy J., 1980. "Analysis of Mechanisms and Robot Arm Manipulators", Wiley, New York, 1980.

[4] Guez A. and Ahmad Z.,1988. "Solution to the Inverse Kinematic Problem in Robotics by Neural Networks", IEEE Int'l Conf. on Neural Networks, San Diego, California, Volume II pp. 617-621,July 1988.

[5] Guo J. and Cherkassky V.,1989. "A Solution to the Inverse Kinematic Problem In Robotics Using Neural Network Processing", IEEE Int'l Joint. Conf. on Neural Networks, Washington D.C., Volume II pp. 299-304, June 1989.

[6] Kohonen T.,1984. "Self-Organization and Associative Memory", Springer Verlag, 1984.
[7] Kuperstein M., and Wang J. ,1990. "Neural controller for adaptive movements with unforseen payloads" IEEE Transactions on Neural Networks, vol 1., No. 1, March 1990.
[8] Miller III, W. T., 1987. "Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm" IEEE J. of Robotics and Automation, vol RA-3, No. 2, April 1987.
[9] Ritter H., Martinetz T., and Schulten K.,1988. "Topology-Conserving Maps for Learning Coordination," Neural Networks, vol. 2, pp. 159-168, 1988.
[10] Ritter H., Martinetz T., and Schulten K.,1990. "Three-Dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm," IEEE Transactions on Neural Networks, Vol. 1, No. 1, pp.131-136, March 1990.

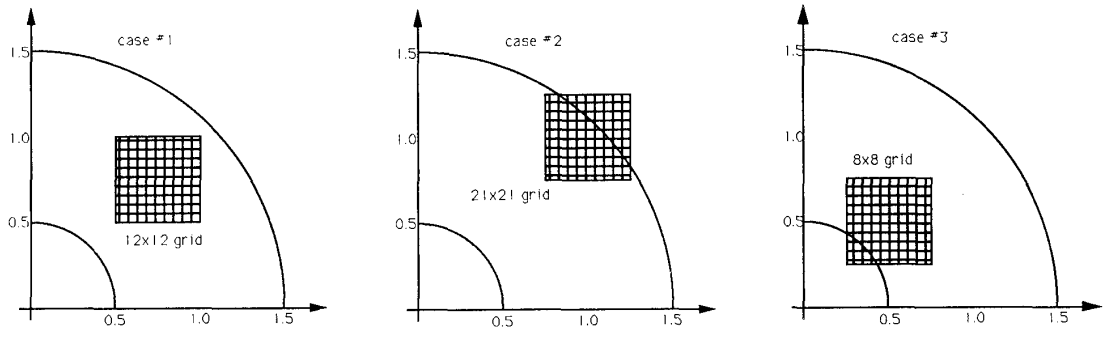

Fig. 3. Test cases in the manipulator's workspace

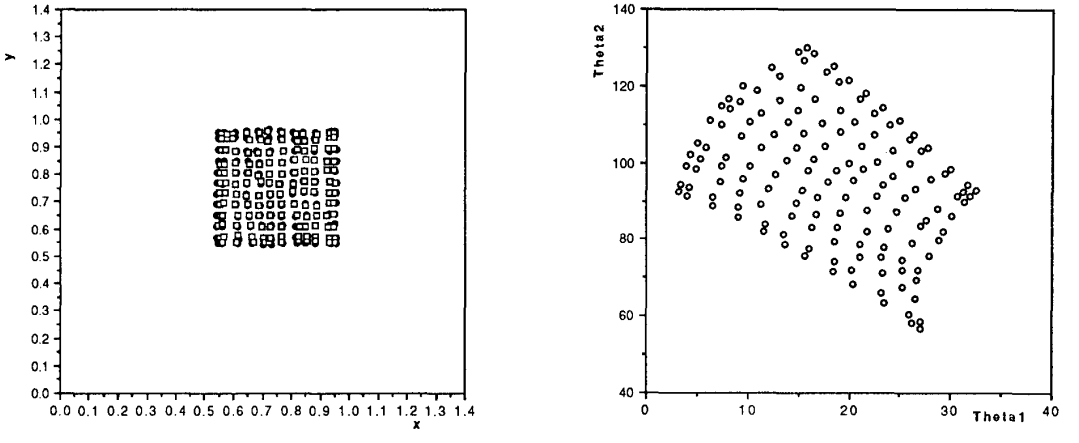

Fig. 4. Results following 12,000 learning steps for 12x12 neuronal grid (case 1). Fig. 4a. Cartesian space representation. Black dots: $\mathbf{w_r}$. White squares: Results after learning $\mathbf{A_r}$. Fig. 4b. The learned results for the joint angles, $\mathbf{Q_r} = (q_{r1}, q_{r2})$ for each of the 12x12 neurons after 12,000 steps.

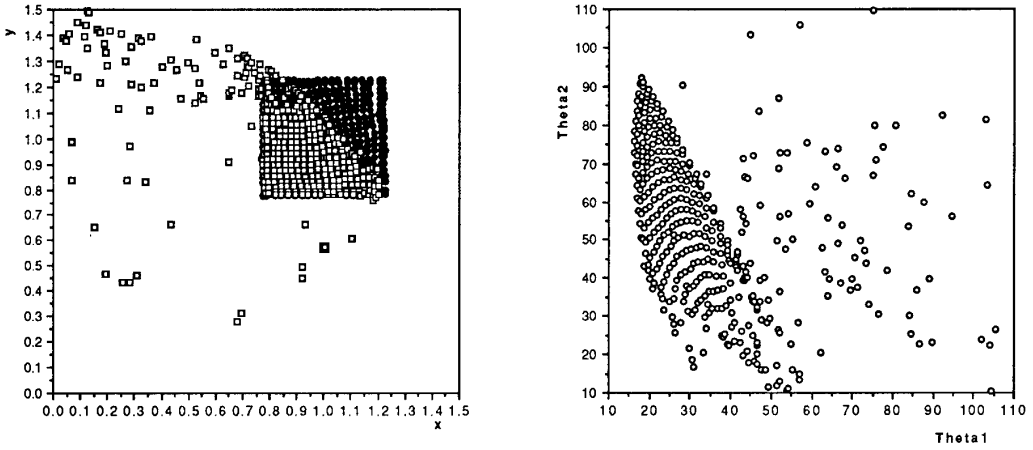

Fig. 5. Results following 20,000 learning steps for 21x21 neuronal grid (case 2). Fig. 5a. Cartesian space representation. Black dots: $\mathbf{w_r}$. White squares: Results after learning $\mathbf{A_r}$. Fig. 5b. The learned results for the joint angles, $\mathbf{Q_r} = (q_{r1}, q_{r2})$ for each of the 21x21 neurons after 20,000 steps.

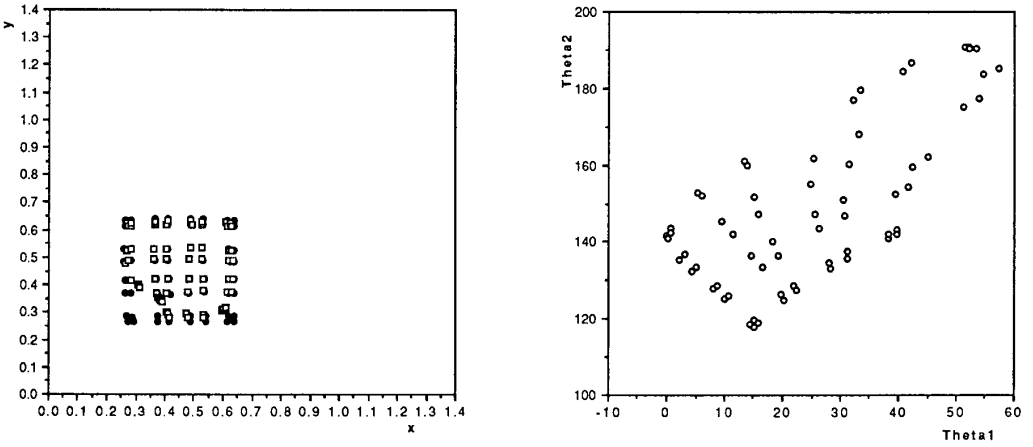

Fig. 6. Results following 20,000 learning steps for 8x8 neuronal grid (case 3). Fig. 6a. Cartesian space representation. Black dots: $\mathbf{w_r}$. White squares: Results after learning $\mathbf{A_r}$. Fig. 6b. The learned results for the joint angles, $\mathbf{Q_r} = (q_{r1}, q_{r2})$ for each of the 8x8 neurons after 20,000 steps.

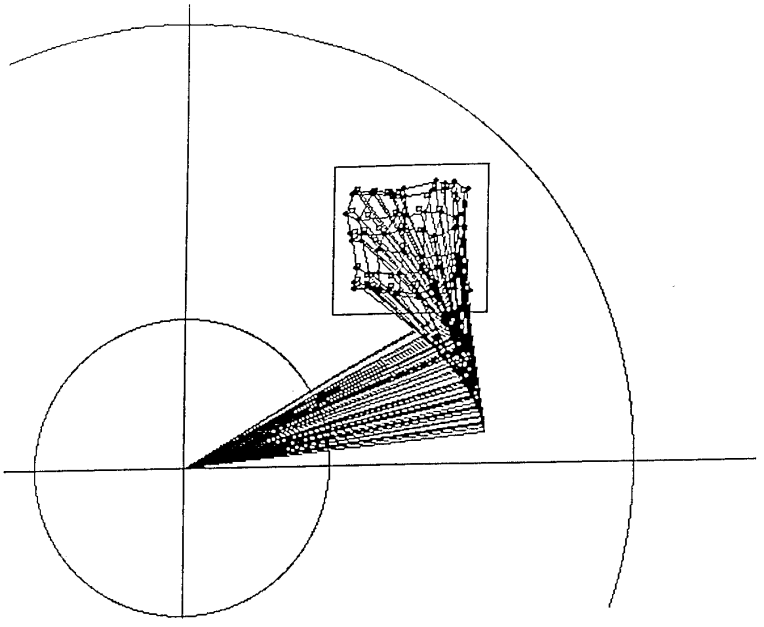

Fig. 7a. Left hand solution for 8x8 grid after 3,000 learning steps.

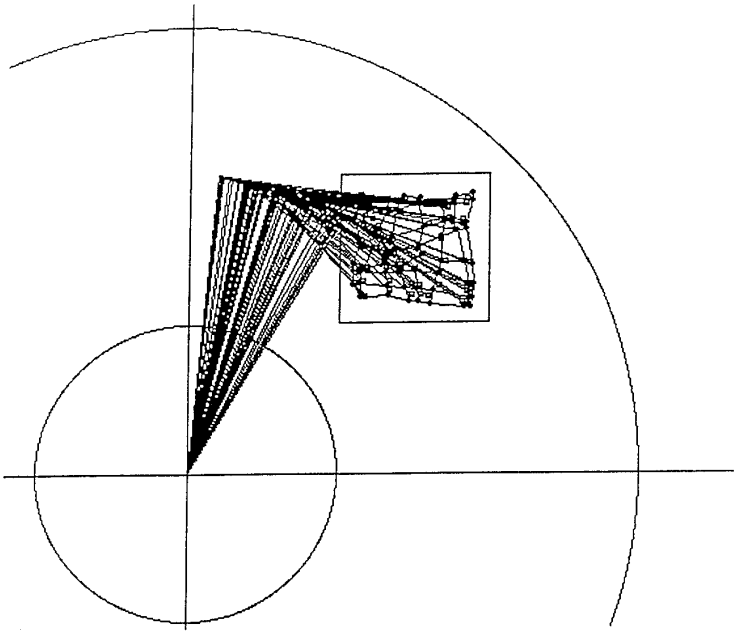

Fig. 7b. Right hand solution for 8x8 grid after 3,000 learning steps.