Regularization of Inverse Kinematics for Redundant Manipulators Using Neural Network Inversions

Bao-Liang Lu [1] and Koji Ito [1,2]
[1]Bio-Mimetic Control Research Center, RIKEN
3-8-31 Rokuban, Atsuta-ku, Nagoya 456, Japan
[2]Dept. of Info. and Computer Sciences, Toyohashi Univ. of Tech.
1-1 Hibarigaoka, Tempaku-cho, Toyohashi 441, Japan
lbl@nagoya.bmc.riken.go.jp; koji@system.tutics.tut.ac.jp

ABSTRACT

This paper presents a new approach to regularizing the inverse kinematics problem for redundant manipulators using neural network inversions. This approach is a four-phase procedure. In the first phase, the configuration space and associated workspace are partitioned into a set of regions. In the second phase, a set of modular neural networks is trained on associated training data sets sampled over these regions to learn the forward kinematic function. In the third phase, the multiple inverse kinematic solutions for a desired end-effector position are obtained by inverting the corresponding modular neural networks. In the fourth phase, an "optimal" inverse kinematic solution is selected from the multiple solutions according to a given criterion. This approach has an important feature in comparison with existing methods, that is, both the inverse kinematic solutions located in the multiple solution branches and the ones that belong to the same solution branch can be found. As a result, better control of the manipulator using the optimum solution than that using an ordinary solution can be achieved. This approach is illustrated with a three-joint planar arm.

1. Introduction

The forward kinematic function for a robot manipulator is a nonlinear mapping, $\boldsymbol{h} : \boldsymbol{Q} \subseteq \mathbf{R}^n \rightarrow \boldsymbol{P} \subseteq \mathbf{R}^m$, which maps a set of joint-angle variables from the configuration space, $\boldsymbol{Q}$, to the workspace, $\boldsymbol{P}$. When $m < n$, the manipulator is called a redundant manipulator. The inverse kinematics problem for redundant manipulators is to find some joint-angle values $\boldsymbol{q} \in \boldsymbol{Q}$ such that $\boldsymbol{h}(\boldsymbol{q})$ is a desired end-effector position $\boldsymbol{p} \in \boldsymbol{P}$. This problem is an ill-posed problem because the inverse kinematic mapping, $\boldsymbol{h}^{-1} : \boldsymbol{P} \subseteq \mathbf{R}^m \rightarrow \boldsymbol{Q} \subseteq \mathbf{R}^n$, is an one-to-many mapping. In general, this problem is locally ill-posed in the sense that it has no unique solution and globally ill-posed because there are multiple solution branches [2], and hence, there is no closed-form direct expression for the inverse kinematic mapping.

In the last few years, several approaches to solving the inverse kinematics problem using neural networks have been proposed [2, 3, 4, 9]. Two popular methods are the direct inverse approach [4] and the distal learning approach [3]. The direct inverse approach learns the inverse kinematic mapping directly using supervised learning algorithms. This approach suffers two main drawbacks that limit its usefulness. First, when the inverse images are nonconvex, the inverse kinematic solution can not be obtained by this approach because any supervised learning algorithm is unable to learn an one-to-many mapping. Second, the multiple solution branch problem is not considered, and hence, only one inverse solution $\boldsymbol{q}$ can be found for a desired end-effector position $\boldsymbol{p}$. The distal learning approach is a two-phase procedure. In the first phase, a network called Net_f is trained to approximate the forward kinematic mapping, $\boldsymbol{Q} \rightarrow \boldsymbol{P}$. After the training is completed, all the parameters of Net_f are fixed. In the second phase, a particular inverse solution is obtained by placing another network called Net_i and Net_f in series and learning an identity mapping across the composite network formed from Net_i and Net_f. Although the distal learning approach can overcome the nonconvex problem encounted by the direct inverse approach, the multiple solution branch problem is still unsolved.

Furthermore, a global regularization approach has been studied in [2]. The basic idea of this approach is that the configuration space is partitioned into regions using unsupervised learning algorithms such that the forward kinematic functions over these regions are invertible, and then the inverse kinematic mapping on each of the partitions is approximated using supervised learning algorithms. But, at the present stage, this approach can only deal with the inverse kinematics problem for 3-link manipulators with one excess degree of freedom.

This paper presents a new approach to regularizing the inverse kinematics problem for redundant manipulators using neural network inversions. This approach is a four-phase procedure. In the first phase, the configuration space and associated workspace are partitioned into a set of regions. In the second phase, a set of modular neural networks is trained on associated training data sets sampled over these regions to learn the forward kinematic function. In the third phase, the multiple inverse kinematic solutions for a desired end-effector position are obtained by inverting the corresponding modular neural networks. In the fourth phase, an "optimal" solution is selected from the multiple solutions according to a given criterion. This approach has an important feature in comparison with existing methods, that is, both the inverse kinematic solutions located in the multiple solution branches and the ones that belong to the same solution branch can be found. As a result, better control of the manipulator using the optimum solution than that using an ordinary solution can be achieved.

2. Neural Network Inversions

The inversion problem for multilayer networks is to find inputs which yield a desired output. There are three common used approaches to inverting multilayer networks in the neural network literature, i.e., the error back-propagation approach [6, 12], the optimization approach [7, 8], and the iterative approach based on the update of input vector [5]. In this paper, the optimization approach is used because many inversions corresponding a desired output can be found by this approach.

A trained three-layer network can be regarded as a mapping from the input space to the output space. In terms of matrix notation, this mapping can be expressed as follows[1]:

$$\begin{aligned} \boldsymbol{x}_2 &= \boldsymbol{f}_2(W_2\,\boldsymbol{x}_1 + \boldsymbol{bias}_2) \\ \boldsymbol{x}_3 &= \boldsymbol{f}_3(W_3\,\boldsymbol{x}_2 + \boldsymbol{bias}_3) \end{aligned} \tag{1}$$

where $\boldsymbol{x}_k = [x_{k1},\, x_{k2},\, \cdots,\, x_{kN_k}]^T \in \mathbf{R}^{N_k}$, $k = 1,2,3$, x_{kj} is the output of the jth unit in the layer k, x_{1j} denotes the input of the jth unit, $W_r = [\boldsymbol{w}_{r1},\, \boldsymbol{w}_{r2}, \cdots, \boldsymbol{w}_{rN_r}]^T \in \mathbf{R}^{N_r \times N_{r-1}}$, $\boldsymbol{w}_{rj} = [w_{rj1},\ w_{rj2},\ \cdots,\ w_{rjN_{r-1}}]$, $j = 1,2,\ \cdots,\ N_r$, w_{kji} is the weight connecting the ith unit in the layer $(k-1)$ to the jth unit in the layer k, $\boldsymbol{f}_r = [f_{r1}, f_{r2}, \cdots, f_{rN_r}]^T$, $f_{kj}(\cdot)$ is the sigmoid activation function of jth unit in the layer k, $\boldsymbol{bias}_r = [\ bias_{r1}, bias_{r2},\ \cdots,\ bias_{rN_r}]^T \in \mathbf{R}^{N_r}$, $r = 2,3$, and $bias_{kj}$ is the bias of the jth unit in the layer k.

For a desired output $\bar{\boldsymbol{x}}_3$, the input $\boldsymbol{x}_1$ which satisfies Eq. (1) is called an inversion. In general, there are an infinite number of inversions corresponding to a desired output. Obviously, finding all of these inversions is impossible in practical computation. A practical strategy is to restrict ourselves to finding some specific inversions. In the optimization approach, the inversion problem is formulated as a nonlinear programming problem [1] as follows:

$$\begin{aligned} &\text{Minimize } g(\boldsymbol{x}_1) \text{ or Maximize } g(\boldsymbol{x}_1) \\ &\text{Subject to} \\ &W_3\,\boldsymbol{f}_2\,(\tilde{\boldsymbol{b}}_2) = \tilde{\boldsymbol{b}}_3 - \boldsymbol{bias}_3 \\ &W_2\,\boldsymbol{x}_1 - \tilde{\boldsymbol{b}}_2 = -\boldsymbol{bias}_2 \\ &\Gamma \le \boldsymbol{x}_1 \le \Theta \end{aligned} \tag{2}$$

where $\tilde{\boldsymbol{b}}_r = \boldsymbol{b}_r + \boldsymbol{bias}_r$ for $r = 2,3$, $\tilde{\boldsymbol{b}}_r \in \mathbf{R}^{N_r}$, b_{kj} is the total net input to the jth unit in the layer k, excluding $bias_{kj}$, $\Gamma = [\gamma_1, \gamma_2, \cdots, \gamma_{N_1}]^T$, $\Theta = [\theta_1, \theta_2, \cdots, \theta_{N_1}]^T$, $\tilde{\boldsymbol{b}}_3 = \boldsymbol{b}_3 + \boldsymbol{bias}_3 = \boldsymbol{f}_3^{-1}(\bar{\boldsymbol{x}}_3)$ is given, $\boldsymbol{x}_1$ and $\tilde{\boldsymbol{b}}_2$ are unknown vectors. The introduction of $\Gamma \le \boldsymbol{x}_1 \le \Theta$ into Eq. (2) is to limit the values of obtained inversions within meaningful ranges. The objective function $g(\boldsymbol{x}_1)$ can take a form: $g(\boldsymbol{x}_1) = \pm x_{1l}$ for $l = 1, 2,\ \cdots,$ or N_1; $g(\boldsymbol{x}_1) = \|\ \boldsymbol{x}_1 - \boldsymbol{c}\ \|^2$, where $\boldsymbol{c} = [c_1,\ c_2,\ \cdots,\ c_{N_1}]^T \in \mathbf{R}^{N_1}$ is a given reference point in the input space; or any one of other optimization criteria. Figure 1 illustrates two kinds of network inversions [8], namely, IMSI(Inversion unilaterally Minimizing or Maximizing Single Input variable) and INSI(Inversion Nearest the Specified Input), in the two-dimensional input space. The IMSIs and INSIs can be obtained by solving the nonlinear programming problem defined by Eq. (2) using the objective function $g(\boldsymbol{x}_1) = x_{1i}$ for $i = 1, 2, \cdots, N_1$, and the objective function $g(\boldsymbol{x}_1) = \|\ \boldsymbol{x}_1 - \boldsymbol{c}\ \|^2$, respectively.

The inversion problem defined by Eq. (2) is a *nonlinear separable programming problem.* Nonlinear separable programming problem refers to a nonlinear programming problem where the objective and the constraint functions can be expressed as a sum of functions, each involving only one variable [1]. An important advantage of the nonlinear separable programming problem is that it can be approximated by a pseudo linear programming problem and solved by a variation of the *simplex* method, a common and efficient technique for solving linear programming problems.

[1]Note that for following the conventional notation in the robotics and neural networks literature, we use two sets of symbols for describing the inverse kinematics problem and neural network inversions, respectively. The relationships among some of the major symbols are as follows: $q \equiv \boldsymbol{x}_1$, $p \equiv \boldsymbol{x}_3$, $n \equiv N_1$, and $m \equiv N_3$.

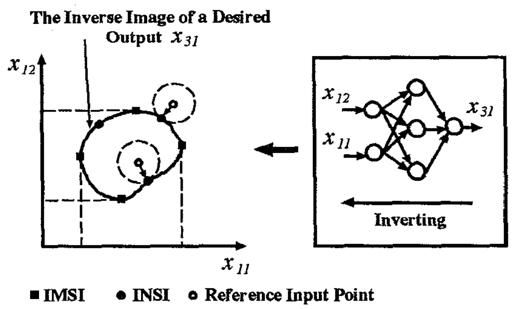

Fig. 1: Illustration of the IMSI and INSI in the two-dimensional input space.

3. Learning Forward Kinematics

The forward kinematic function can be approximated by a monolithic multilayer neural network. This strategy is commonly used for approximation of inverse kinematics in the robotics literature. However, the monolithic structure has two drawbacks: (a) to approximate the forward kinematic function of a manipulator with large excess degrees of freedom requires long training time from the point of view of learning a function, and (b) to invert the network becomes difficult because the large-scale network increases complexity of the inversion problem from the point of view of inverting a network. In order to overcome the above disadvantages, a modular network scheme is introduced in this paper.

To implement the modular scheme, we need to decompose the learning task into subtasks which can be learned by individual modular networks. For the forward kinematic function approximation problem, this can be achieved by partitioning the configuration space into a set of regions. We assume that the configuration space is a convex polyhedron in $\mathbf{R}^n$ [10]. For the simplest special case, it can be expressed as the following form:

$$\boldsymbol{Q} \equiv \{\boldsymbol{q} \in \mathbf{R}^n \mid \boldsymbol{q}^{\min} \leq \boldsymbol{q} \leq \boldsymbol{q}^{\max}\} \qquad (3)$$

where $\boldsymbol{q}^{\min}$ and $\boldsymbol{q}^{\max}$ are constant vectors and represent lower and upper joint-angle limits, respectively. In this case, the configuration space $\boldsymbol{Q}$ can be easily partitioned into τ ($\tau \geq 1$) smaller convex polyhedrons as follows:

$$\boldsymbol{Q}_i \equiv \{\boldsymbol{q}_i \in \mathbf{R}^n \mid \boldsymbol{q}_i{}^{\min} \leq \boldsymbol{q}_i \leq \boldsymbol{q}_i{}^{\max}\} \quad (4)$$
$$\text{for } i = 1,\ 2,\ \cdots,\ \tau$$

where $\boldsymbol{q}_i{}^{\min}$ and $\boldsymbol{q}_i{}^{\max}$ are also constant vectors and represent lower and upper joint-angle limits in the ith region $\boldsymbol{Q}_i$, respectively.

Based on the above discussion, the forward kinematic function can be learned by individual modular neural networks according to the following algorithm:

Step 1 : Partition the configuration space $\boldsymbol{Q}$ into τ regions $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, $\cdots$, $\boldsymbol{Q}_\tau$ in a uniform grid or in a non-uniform grid.

Step 2 : Gather the input-output training data sets T_1, T_2, $\cdots$, T_τ by sampling the corresponding overlapping regions $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, $\cdots$, $\boldsymbol{Q}_\tau$, and identifying the associated points in the workspace.

Step 3 : Memorize the ranges of unnormalized desired outputs[2] in T_1, T_2, $\cdots$, T_τ, in order to select modular networks for inverting.

Step 4 : Train the modular neural network MN_1, MN_2, $\cdots$, MN_τ on T_1, T_2, $\cdots$, T_τ, respectively.

It is important to emphasize that, besides overcoming the drawbacks of the monolithic structure mentioned above, the modular network scheme can regularize the ill-posed inverse kinematics problem globally to certain extent. Although there is no theoretical guarantee that all multiple solution branches can be partitioned by dividing the configuration space into a set of regions, typically, the smaller the region is divided, the more complete the global regularization can be achieved. Figure 2 shows the modular network scheme for learning the forward kinematic function.

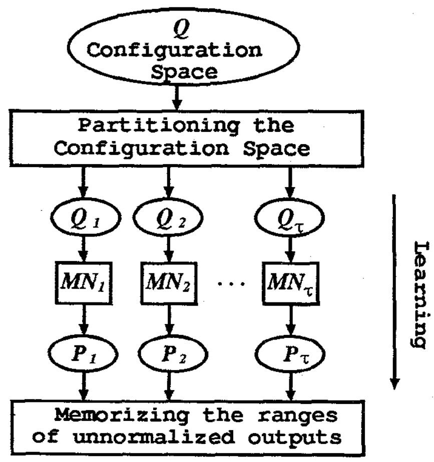

Fig. 2: The modular network scheme for learning the forward kinematic function.

4. Regularization Using Inversions

If a forward kinematic function is approximated precisely by a multilayer network, then solving

[2]Since all of the training input and output data are normalized between 0 and 1 before training, we can only distinguish the output ranges by using unnormalized data after training.

the inverse kinematics problem is equivalent to inverting the multilayer network. Suppose a desired end-effector position $\boldsymbol{p}$ is a point within the workspace $\boldsymbol{P}$. To compute the inverse kinematic solutions associated with $\boldsymbol{p}$, we must determine which modular networks need to be inverted. However, it is difficult for us to make a precise choice because, in general, $\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_\tau$, the images of $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \cdots, \boldsymbol{Q}_\tau$, are non-convex sets and there is no closed-form expression. In this paper, we use a simple approach to tackling this problem, that is, whether $\boldsymbol{p}$ is within $\boldsymbol{P}_i$ is judged according to the range of unnormalized training outputs in T_i. For example, if $\boldsymbol{p} = (0.65,\ 0.0)$ and the ranges of unnormalized training outputs in T_1 and T_2 are $\{(0.01,\ 0.70),\ (-0.40,\ 0.60)\}$ and $\{(-0.55,\ 0.53),\ (0.16,\ 0.70)\}$, respectively (see Table 1), then only MN_1 is selected because $\boldsymbol{p}$ is within the range of T_1, while $\boldsymbol{p}$ is not included in the range of T_2. This is an approximate approach, and therefore, more number of the modular networks than those actually required may be selected and there may exist no inverse solution for some modular networks that are selected incorrectly. For a desired end-effector position $\boldsymbol{p}$, σ $(1 \leq \sigma \leq \tau)$ modular networks may be selected for inverting because some of $\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_\tau$ overlap each other.

The goal of the proposed approach is to find different inverse kinematic solutions $\boldsymbol{q}$ as many as possible for a desired end-effector position $\boldsymbol{p}$. The multiple solutions to the inverse kinematics problem can be obtained by the following algorithm:

Step 1: Determine which modular network needs to be inverted.
Step 2: Select objective function $g(\boldsymbol{q})^3$ in Eq. (2).
Step 3: Compute an inverse kinematic solution by solving the separable nonlinear programming problem defined by Eq. (2).
Step 4: Repeat Steps 2 and 3 until a desired number of inverse solutions are obtained.

Figure 3 illustrates the scheme for computing multiple inverse kinematic solutions by inverting modular neural networks.

After the multiple inverse solutions have been obtained, we must select an "optimum" one from them. The "optimum" inverse solution refers to the best joint angles $\boldsymbol{q}$ in obtained multiple inverse kinematic solutions for a desired end-effector position $\boldsymbol{p}$. The criterion for choosing an "optimum" inverse solution is largely dependent on the requirement of a manipulator, and it is difficult to give a general criterion. We will discuss how to choose an optimum inverse solution through a simple example.

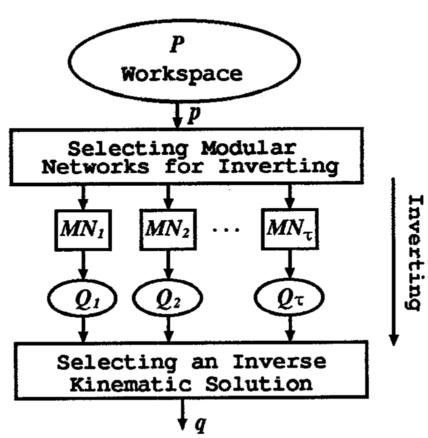

Fig. 3: The scheme for computing multiple inverse kinematic solutions.

Consider, for example, a three-joint planar arm as shown in Figure 4. The workspace contains an obstacle. If the manipulator needs to move from position $\boldsymbol{p}'$ to position $\boldsymbol{p}$, we must find the corresponding $\boldsymbol{q}$. According to Eq. (2), we can obtain many inverse kinematic solutions by setting the objective function in the forms: Minimize q_i or Maximize q_i for $i = 1, 2, or\, 3$. For example, two different inverse solutions are illustrated in Figure 4 as the *elbow-up* and *elbow-down* solutions. In this case, the movement may not be free of collisions between the manipulator and the obstacle. In order to avoid the obstacle we prefer to select the *elbow-up* solution as an "optimal" one since the *elbow-up* solution reduces the chance of a collision between the links of the manipulator and the obstacle resting on the workspace.

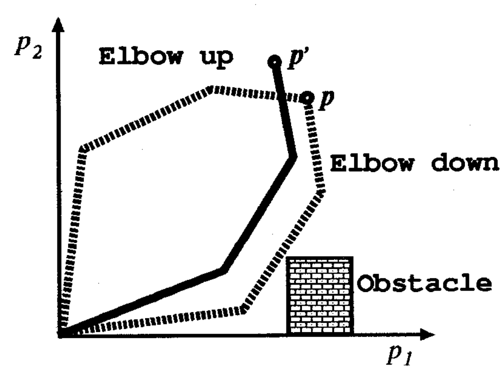

Fig. 4: Two different inverse kinematic solutions for a redundant manipulator.

[3] Note that for following the conventional notation in the robotics and neural networks literature, we use two sets of symbols for describing the inverse kinematics problem and neural network inversions, respectively. The relationships among some of the major symbols are as follows: $q \equiv x_1$, $p \equiv x_3$, $n \equiv N_1$, and $m \equiv N_3$.

5. Simulation Results

In order to demonstrate the proposed approach, the simulations are carried out on a three-joint planar arm as shown in Figure 5. The configuration of the arm is characterized by the three joint angles, q_1, q_2, and q_3, and the corresponding pair of Cartesian variables p_1 and p_2. Without loss of generality and for simplicity of illustration, the precise analytic forward kinematic function of the arm is used for generating training input-output data. It is expressed as follows:

$$\begin{aligned} p_1 &= L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) + \\ &\quad L_3 \cos(q_1 + q_2 + q_3) \\ p_2 &= L_1 \sin(q_1) + L_2 \sin(q_1 + q_2) + \\ &\quad L_3 \sin(q_1 + q_2 + q_3) \end{aligned} \tag{5}$$

where L_1, L_2, and L_3 are the manipulator link lengths. We set $L_1 = 0.3$, $L_2 = 0.25$, and $L_3 = 0.15$, and restrict the motion of the joints q_1, q_2, and q_3 to the intervals $[-\pi/6,\ 2\pi/3]$, $[0,\ 5\pi/6]$, and $[-\pi/6,\ \pi/6]$, respectively.

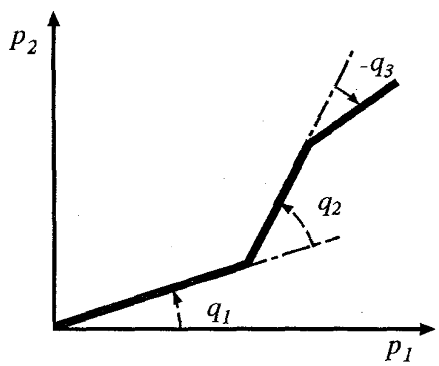

Fig. 5: A three-joint planar arm.

The configuration space $\boldsymbol{Q}$ is divided into 8 overlapping regions $\hat{\boldsymbol{Q}}_1$, $\hat{\boldsymbol{Q}}_2$, $\cdots$, $\hat{\boldsymbol{Q}}_8$, via the grid pionts $(-\pi/6,\ 3\pi/12,\ 2\pi/3)$, $(0,\ 5\pi/12,\ 5\pi/6)$, and $(-\pi/6,\ 0,\ \pi/6)$. For example, $\hat{\boldsymbol{Q}}_1$ is partitioned by the intervals $[-\pi/6,\ 3\pi/12]$, $[0,\ 5\pi/12]$, and $[-\pi/6,\ 0]$. Over each of the regions, a set of 216 ($6 \times 6 \times 6$) training input-output data is sampled using Eq. (5) in a non-uniform grid. Table 1 shows the ranges of unnormalized outputs of T_1, T_2, $\cdots$, T_8, respectively. Eight modular networks are used for approximating the forward kinematic function. Each of them is a three-layer network with 3 input, 10 hidden and 2 output units, and is trained by the backpropagation learning algorithm [11].

To compute multiple inverse kinematic solutions, we select the objective function $g(\boldsymbol{q})$ in Eqs. (2) as: $g(\boldsymbol{q}) = \pm q_i$ for $i = 1,\ 2$, or 3. This objective function allows us to minimize or maximize the movement of the ith link. Let us compute the inverse kinematic solutions for the desired end-effector position $\boldsymbol{p} = (-0.4,\ 0.2)$. Since the desired end-effector position $\boldsymbol{p}$ is located in the ranges of the training outputs of T_2, T_4, T_6 and T_8, MN_2, MN_4, MN_6, and MN_8 are selected for inverting. However, only four distinct inverse solutions, which are shown in Table 2 and illustrated in Figure 6, are obtained by inverting MN_4 and MN_8, and there exists no any inverse solution in MN_2 and MN_6. The reason for this situation is that $\boldsymbol{p} = (-0.4,\ 0.2)$ is within the training output ranges of T_2 and T_6, but it doesn't locate in the actual output areas of MN_2 and MN_6.

Table: 1: The ranges of unnormalized outputs in T_i for $i = 1,\ 2,\ \cdots,\ 8$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$p_1^{\min}$	.01	-.55	-.20	-.56	-.06	-.57	-.19	-.58
$p_1^{\max}$	.70	.53	.61	.18	.70	.53	.59	.02
$p_2^{\min}$	-.40	.16	.13	-.16	-.35	.11	.04	-.17
$p_2^{\max}$	.60	.70	.57	.58	.60	.69	.52	.59

From the above simulation results, we see that the inverse kinematic solutions in multiple solution branches can be found by inverting the corresponding modular networks. This demonstrates that our approach can regularize the inverse kinematics problem globally, and furthermore, distinct solutions in the same solution branch can also be obtained by solving the optimization problem defined in Eq. (2) with different objective functions.

After the multiple inverse kinematic solution have been obtained, we can select an optimum solution from them according to requirement of the manipulator. For example, in order to avoid the obstacle resting on the workspace, the *elbow-up* solution $\boldsymbol{q} = (96.0,\ 87.9,\ 29.9)$ as shown in Figure 6 is selected as an optimum solution from four inverse solutions as shown in Table 2.

6. Conclusion

In this paper, we have presented a new approach to solving the inverse kinematics problem for redundant manipulators. This approach is based on modular neural network scheme and network inversion techniques. This approach has an important feature in comparison with existing methods, that is, both the inverse kinematic solutions located in multiple solution branches and ones that belong to the same solution branch can be found by inverting the corresponding modular neural networks. Therefore, an optimum inverse kinematic solution can be found and better control of the manipulator can be achieved. As future work we will develop efficient

Table: 2: The Inverse Kinematic Solutions for $p_1 = -0.4$ and $p_2 = 0.2$, and the Corresponding Actual Positions

No.		Inverse Kinematic Solutions			Positions	
		q_1	q_2	q_3	p_1	p_2
MN_4	Min(q_1)	92.3	101.6	0.0	-0.400	0.204
	Max(q_1)	95.5	110.7	-29.9	-0.403	0.198
MN_8	Min(q_1)	93.4	100.9	0.0	-0.405	0.201
	Max(q_1)	96.0	87.9	29.9	-0.405	0.198

approach to selecting modular networks for inverting and perform simulations on manipulators with large excess degrees of freedom.

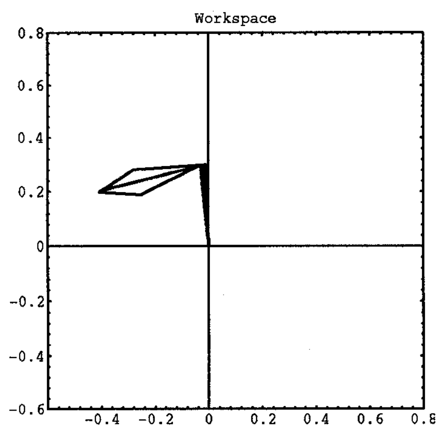

Fig. 6: Four different inverse kinematic solutions obtained by inverting MN_4 and MN_8 for the desired end-effector position $p_1 = -0.4$ and $p_2 = 0.2$. Note that two of solutions are quite near and overlapped in the figure.

References

[1] Bazaraa, M. S. and Shetty, C. W. : *Nonlinear Programming Theory and Algorithms*, Jonn Wiley and Sons, 1979.

[2] Demers, D. E. : *Learning to Invert Many-to-one Mappings*, Ph. D. thesis, University of California, San Diego, 1993.

[3] Jordan M. I. : "Forward models: supervised learning with a distal teacher", *Cognitive Science*, vol. 16, pp. 307-354, 1992.

[4] Kuperstein M. : "Adaptive visual-motor coordination in multijoint robots using parallel architecture", *Proc. of 1987 IEEE Int. Conf. Robotics and Automation*, pp. 1595-1602, Raleigh, North Carolina, 1987.

[5] Lee, S. and Kil, R. M. : "Inverse mapping of continuous functions using local and global information", *IEEE Trans. Neural Networks*, vol. 5, no. 3, pp. 409-423, 1994

[6] Linden, A. and Kindermann, J. : "Inversion of Multilayer Nets", *Proc. of International Joint Conference on Neural Networks*, Washington, vol. 2, pp. 425-430, 1989.

[7] Lu, B. L., Kita, H. and Nishikawa, Y. : "Inversion of feedforward neural networks by a separable programming", *Proc. of World Congress on Neural Networks*, vol. 4, pp. 415-420, Portland, OR, 1993.

[8] Lu, B. L. : *Architectures, Learning and Inversion Algorithms for Multilayer Neural Networks*, Ph. D. thesis, Dept. of Electrical Engineering, Kyoto University, 1994.

[9] Ritter, H. J., Martinetz, T. M. and Schulten, K. J. : "Topology-conserving maps for learning visuo-motor-coordination", *Neural Networks*, vol. 2, pp. 159-168, 1989.

[10] Schilling, R. J.: *Fundamentals of Robotics: Analysis and Control*, Prentice Hall, 1994.

[11] Rumelhart, D. E. and McClelland, J. L., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, The MIT Press, 1986.

[12] Williams, R. J. : "Inverting a connectionist network mapping by backpropagation of error", *Proc. of 8th Annual Conference of th Cognitive Science Society*, Lawrence-Erlbaum, pp. 859-865, 1986.