INT J COMPUT COMMUN, ISSN 1841-9836
Vol.7 (2012), No. 3 (September), pp. 459-472

# Inverse Kinematics Solution for Robot Manipulator based on Neural Network under Joint Subspace

Y. Feng, W. Yao-nan, Y. Yi-min

**Yin Feng, Wang Yao-nan, Yang Yi-min**
The College of Electrical and Information Engineering
Hunan University, Changsha, Hunan Province 410082, P.R.China
E-mail: yinfeng83@126.com, yaonan@hnu.cn, yimin-yang@126.com

**Abstract:** Neural networks with their inherent learning ability have been widely applied to solve the robot manipulator inverse kinematics problems. However, there are still two open problems: (1) without knowing inverse kinematic expressions, these solutions have the difficulty of how to collect training sets, and (2) the gradient-based learning algorithms can cause a very slow training process, especially for a complex configuration, or a large set of training data. Unlike these traditional implementations, the proposed metho trains neural network in joint subspace which can be easily calculated with electromagnetism-like method. The kinematics equation and its inverse are one-to-one mapping within the subspace. Thus the constrained training sets can be easily collected by forward kinematics relations. For issue 2, this paper uses a novel learning algorithm called extreme learning machine (ELM) which randomly choose the input weights and analytically determines the output weights of the single hidden layer feedforward neural networks (SLFNs). In theory, this algorithm tends to provide the best generalization performance at extremely fast learning speed. The results show that the proposed approach has not only greatly reduced the computation time but also improved the precision.
**Keywords:** Inverse kinematics, neural network, extreme learning machine.

## 1 Introduction

The inverse kinematics (IK) problem for a serial-chain manipulator is to find the values of the joint positions given the position and orientation of the end-effector relative to the base. There are many solutions to solve the inverse kinematics problem [1] , such as geometric, algebraic, and numerical iterative methods. In particular, some of the most popular methods are mainly based on inversion of the mapping established between joint space and task space by the Jacobian matrix. This solution uses numerical iteration to invert the forward kinematic Jacobian matrix and does not always guarantee to produce all possible inverse kinematics solutions. The artificial neural network, which has significant flexibility and learning ability, has been used in the inverse kinematics problem. One solution followed a closed-loop control scheme where a neural network is used to directly learn the nonlinear relationship between the displacement in the workspace and control signal in the joint angle space to achieve a desired position([2] and [3]) . Other schemes used neural networks to learn a mapping function from the world space to joint space. Although there are various neural networks, the multi-layer perceptron network (MLPN) and the radial basis function network (RBFN) are the most popular neural network applied to functional approximation problems.

In [4], the effects of structural parameters, iteration steps and different numbers of training points on the performance of the inverse kinematics approximation were investigated. The results showed that a more complex MLPN configuration is likely to produce a more accurate inverse kinematics approximation. However, it also leads to the number of iterations increasing significantly to satisfy the required training goal. Similarly, the neural networkss generalization ability seems to be improved when the number of training sets is increased. However, if the numbers of hidden neurons or training sets are too large,

the training process can not even converge to an expected error goal in some cases. In [5], an MLPN with various structures of the input layer were proposed to solve the inverse kinematics problem of a 6 DOF manipulator. Three different forms representing the orientation of the end-effector with respect to the base were defined: a 33 rotation matrix, a set of 3 Euler angles and one angle and a 13 unit vector. Another solution combining an MLPN and a lookup table to solve an inverse kinematics problem of a redundant manipulator was proposed in [6]. Although the use of MLPN in the inverse kinematics problem has a greater extent, there have some significant disadvantages. For example, there is no reasonable mechanism to select a suitable network configuration relating to the system characteristics represented by training sets. In addition, training MLPN using the back-error propagation algorithm is complex and slow. For a complex MLPN structure required for a complex configuration manipulator, or a large set of training data, the training process is slow to converge to a specific goal. Therefore, trends towards using RBFN which are conceptually simpler and possess the ability to model any nonlinear function conveniently have become more popular.

In [7], a variety of network configurations based on RBFN were developed to explore the effect of various network configurations on the performance of the network. In [8], a novel architecture of RBFN with two hidden layers was developed for a inverse kinematics problem of a 3-link manipulator. A fusion approach was proposed in [9]. The proposed approach used RBFN for prediction of incremental joint angles which in turn were transformed into absolute joint angles with the assistance of forward kinematics relations. Another RBFN-based method was presented in [10]. It developed a structure of six parallel RBFN, each of which consists of six inputs which represent a location of the end-effector and one output as the joint angle. Thus, the group of six parallel RBFN (one for each joint angle) could perform an inverse kinematics approximation. In addition, some hybrid techniques made use of neural networks along with expert system [11], fuzzy logic [12] and genetic algorithm [13] for solving the inverse kinematics. Though these intelligence approaches can be applied for two or three DOF planar robots, they often demand high performance computing systems and complex computer programming for complex robotic system.

Traditionally, all the parameters (weights and biases) of the feedforward networks need to be turned. For past decades gradient descent-based methods have mainly been used in various learning algorithms. It is clear that the learning process often needs many training patterns and times to cover the entire workspace. Thus, it is not surprising to see that it may take several hours and several days to train neural networks to solve the inverse kinematics. Unlike these traditional implementations, this paper uses a novel learning algorithm called extreme learning machine (ELM) for single hidden layer feedforward neural networks (SLFN) which randomly chooses the input weights and biases, and analytically determines the output weights of SLFN [14,15]. In theory, this algorithm tends to provide the best generalization performance at extremely fast learning speed.

Another issue of concern for solving the inverse kinematics using neural networks is the training data sets. As we know, the joint space of the robot can be considered as an inverse image of the Cartesian space and vice versa. Thus, the forward kinematics can be assumed to be an inverse image of inverse kinematics and vice versa [9]. The pose P can be used as an input and the corresponding joint angle Q as the output for the neural network training data. In other words, Q$\longrightarrow$P relationship is used while generating the data whereas P$\longrightarrow$Q mapping is done while training the neural network. Usually, the inverse kinematics problems have multiple solutions. For example, the PUMA 560 robot has at most eight solutions when there are no joint limits imposed. Hence, the inverse kinematics equation is one-to-many mapping.Unfortunately,the neural network can not match the actual output with the desired output. So the learning error of neural network is hard to be calculated when training. An effective solution is that the training sets are constrained to only one solution set so that the one-to-one mapping can be achieved. For simple structure, such as two-link planar manipulator, the training sets can be collected based on the inverse kinematics equation which only consist of either the positive or negative solution. However, this solution has the difficulty of how to collect constrained data without knowing the inverse kinematic

expressions of the complex robotic system. The present work attempts to resolve this crucial issue by using a novel heuristic algorithm, called electromagnetism-like method (EM)[16,17], for determining a joint subspace which includes one and only one inverse kinematics solution for a given pose. For convenience's sake, a graphic depiction of the proposed method is illustrated by using a 2D example, as shown in figure 1.

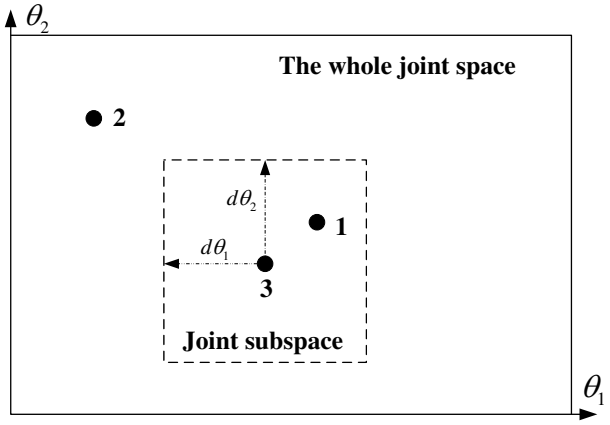


Figure 1: An illustration of the proposed algorithm. The point 1 and 2 is true solution and point 3 is an approximate solution

In Figure 1, assume that there are two inverse solutions in the whole space. One approximate solution is denoted by $(\theta_1^*, \theta_2^*)$ . If we select appropriate $d\theta_1$ and $d\theta_2$ such that $\theta_1 \in [\theta_1^* - d\theta_1, \theta_1^* + d\theta_1]$ and $d\theta_2 \in [\theta_2^* - d\theta_2, \theta_2^* + d\theta_2]$, this joint subspace includes just one true solution point 1. Based on this, the data required for training of neural network is proposed to be derived from the joint subspace with the forward kinematics relations instead of deriving a set complex inverse kinematics equations from the whole joint space. Then the true solution point 1 can be approached by using the trained network. The proposed method can be summarized as follows:

1. Given a desired coordinate of position and orientation of the end-effector, making use of EM to calculate an approximate solution near to one true solution.

2. Specify appropriate value of $d\theta_k$ such that $\theta_k \in [\theta_k^* - d\theta_k, \theta_k^* + d\theta_k](k = 1, 2, \cdots, n)$, where $n$ is the number of joint and $\theta_k^*$ is the approximate solution of the $k$ joint variable calculated with EM in step 1. For the sake of simplicity, all $d\theta$ can be set to be the same value.

3. Collect the training sets from the joint subspace, and train the neural network with ELM.

For a new coordinate of position and orientation of the end-effector, the neural networks usually need to be retrained following the steps above. Fortunately, our results show that the training process is very fast. Thus, the retraining procedure appears to be acceptable.

# 2 Calculation of joint subspace with EM

## 2.1 Problem formulation

As shown in Figure 2, the desired position vector and orientation matrix of a manipulator end-effector are denoted by: $\boldsymbol{P_d}$ and $[\boldsymbol{R_d}] = [\boldsymbol{d_1}, d_2, \boldsymbol{d_3}]$, where $\boldsymbol{d_j}$ ($j$=1, 2, 3) are unit vectors along the $\mathbf{x_d}, \mathbf{y_d}, \mathbf{z_d}$ axes. $\boldsymbol{P_h}$ is the current position vector of the end-effector. The current orientation matrix is defined by : $[\boldsymbol{R_h}] = [\boldsymbol{h_1}, \boldsymbol{h_2}, \boldsymbol{h_3}]$, where $\boldsymbol{h_j}$ ($j$=1,2,3) are unit vectors along the $\mathbf{x_h}, \mathbf{y_h}, \mathbf{z_h}$ axes and the joint variables are denoted by the n×1 vector, $\theta = [\theta_1, \theta_2, \cdots, \theta_n]^{\mathrm{T}}$.

The error between the current and the desired locations of the end-effector can be described by the following functions [18]:

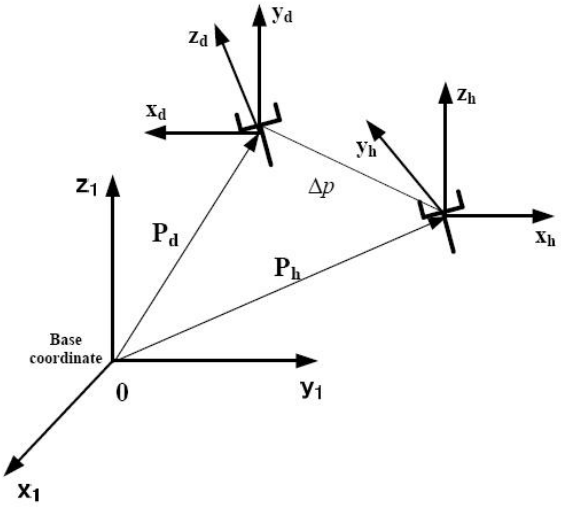


Figure 2: The current and the desired end-effector configurations

Position error:

$$\Delta p(\theta) = \| \boldsymbol{P_d} - \boldsymbol{P_h}(\theta) \| \tag{1}$$

Orientation error:

$$\Delta o(\theta) = \sum_{j=1}^{3} (\boldsymbol{d}_j \cdot \boldsymbol{h}_j(\theta) - 1)^2 \tag{2}$$

The total error:

$$e(\theta) = \Delta p(\theta) + \Delta o(\theta) \tag{3}$$

Where $(\cdot)$ denotes the vector dot product. Furthermore, the total error can be chosen to be a weighted sum of the position and orientation components:

$$e(\theta) = w_p \Delta p(\theta) + w_o \Delta o(\theta) \tag{4}$$

Where $w_p$ and $w_o$ are weighting factors assigned to position and orientation, respectively, such that $w_p + w_o = 1$. Now the inverse kinematics problem is to find a solution $\theta^*$, such that $e(\theta^*) \leq \varepsilon(\varepsilon \longrightarrow 0)$.It is clear that this problem can be transformed into the following minimization problem:

$$\min e(\theta) \quad \text{s.t.} \quad \theta \in \Re^{\mathrm{n}} | \mathrm{l_k} \leq \theta_{\mathrm{k}} \leq \mathrm{u_k}, \mathrm{i} = 1, 2, \cdots, \mathrm{n} \tag{5}$$

## 2.2 Brief of Electromagnetism-like method (EM)

To solve the problem in (5), the general scheme of EM is given by following procedures: Initialize, local search, calculation of charge and total force vector and movement according to the total force.

### Initialization

The procedure initialization is used to sample $m$ points, $\theta^1, \cdots, \theta^m$, randomly from the feasible domain of the joint variables, where $\theta^i = [\theta_1^i, \cdots, \theta_n^i](i = 1, \cdots, m)$. The procedure uniform sampling can be determined by following

$$\theta_k^i = l_k + rand \cdot (u_k - l_k) \qquad k = 1, 2, \cdots, n \tag{6}$$

The procedure ends with $m$ points identified, and the point that has the best function value is stored in $\theta^{best}$.

**Local search**

The local search procedure is used to gather the local information and improve the current solutions. It can be applied to one or many points for local refinement per iteration. The selection of these two procedures, does not affect the convergence result.

**Calculation of charge and total force vector**

The charges of the points are calculated according to their objective function values, and the charge of each point is not constant and changes from iteration to iteration. The charge of the $i$th point, $q^i$, is evaluated as following

$$q^i = \mathbf{exp}[-n\frac{(e(\theta^i) - e(\theta^{best}))}{\sum_{k=1}^{m}(e(\theta^k) - e(\theta^{best}))}], i = 1, 2, \cdots, m \tag{7}$$

In this way, points that have better objective values possess higher charges. Notice that, unlike electrical charges, no signs are attached to the charge of an individual point in (7). Instead, the direction of a particular force between two points is decided after comparing their objective function values. Hence, the total force $\boldsymbol{F_i}$ exerted on point $i$ is computed by the following equation

$$\boldsymbol{F}^i = \begin{cases} \sum_{j\neq i}^{m}(\theta^j - \theta^i)\frac{q^j q^i}{\| \theta^j - \theta^i \|^2}, & if \quad e(\theta^j) < e(\theta^i) \\ \sum_{j\neq i}^{m}(\theta^i - \theta^j)\frac{q^j q^i}{\| \theta^j - \theta^i \|^2}, & others \end{cases} \tag{8}$$

According to (8), the point that has a better objective function value attracts the other one. Contrarily, the point with worse objective function value repels the other. Since $\theta^{best}$ has the minimum objective function value, it acts as an absolute point of attraction. Then it attracts all other points in the population to better region.

**Movement according to the total force**

After evaluating the total force vector $\boldsymbol{F}^i$, the point $i$ is moved in the direction of the force by a random step length in (8). Here the random step length $\lambda$ is assumed to be uniformly distributed between 0 and 1.

$$\theta^i = \theta^i + \lambda\frac{\boldsymbol{F}^i}{\|\boldsymbol{F}^i\|}\boldsymbol{R}_{NG}, \quad i = 1, 2, \cdots, m \tag{9}$$

In (9), $\boldsymbol{R}_{NG}$ is a vector whose components denote the allowed feasible movement toward the upper bound $u_k$ or the lower bound $l_k$ of the joint variables.

After finishing the above procedures, the positions of points are updated and we have finished one iteration calculation of EM. Take the Figure 3 for example. There are three particles and their own objective values are 15, 10 and 5, respectively. Because particle 1 is worse than particle 3 while particle 2 is better than particle 3, particle 1 represents a repulsion force which is F13 and particle 2 encourages particle 3 that moves to the neighborhood region of particle 2. Consequently, particle 3 moves along with the total force F.

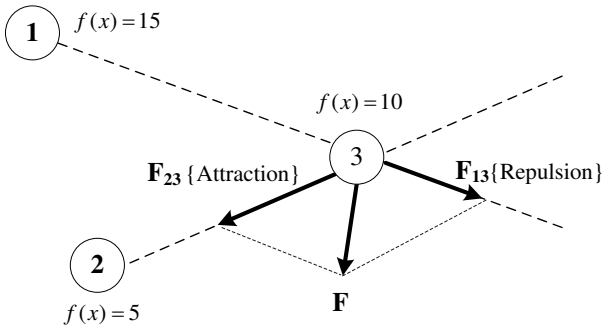


Figure 3: An example of attract-repulse effect on particle number 3

### 2.3 Performance evaluation of EM in solving the inverse kinematics

This example is used to examine the precision of the approximate solution calculated by EM, which directly impact the choice of the interval width $d\theta$. The robot structure for this example is based on PUMA 560. The link parameters are given in Table 1.

Table 1 The Link parameters of the PUMA 560 Robot

| Joint | Link length (m) | Twist angle (degree) | Offset length (m) | Joint limitations (degree) |
|---|---|---|---|---|
| 1 | 0 | -90 | 0.6604 | $[-160, 160]$ |
| 2 | 0.4320 | 0 | 0.2000 | $[-225, 045]$ |
| 3 | 0 | 90 | -0.0505 | $[-045, 225]$ |
| 4 | 0 | -90 | 0.4320 | $[-110, 170]$ |
| 5 | 0 | 90 | 0.0000 | $[-100, 100]$ |
| 6 | 0 | 0 | 0.0565 | $[-266, 266]$ |

The desired configuration of the end-effector is given by: Pd=[0.7433, 0.3111, 0.7883] (m), and d1= [-0.6366, 0.7712, -0.0084], d2= [0.0227, 0.0296, 0.9993], d3 = [0.7709, 0.6359, -0.0364]. Note that there are multiple solutions within the joint limitations shown in table 1. For the sake of simplicity, the joint 1 and 3 limitations are rearranged into [-120,160] and [-45,120], respectively. For the given coordinates of the end-effector, it corresponds to an exact solution of $\theta$ = [10 20 30 40 50 60](degree) within the adjusted joint limitations. Then the error between an approximate solution and the true solution can be easily calculated. The stopping criterion for EM is defined by $\varepsilon$ =0.01. In other words, stop calculation when the total error (see (3)) is less than $\varepsilon$. In this example, 100 trials have been conducted for EM and the maximum absolute error (absolute value) at each joint angle is shown in Figure 4.

From Figure 4, the widths of the joint limitations are set as about 20° at least, i.e.$d\theta$ = 10°, which can guarantee that the one-to-one mapping is achieved. It should be noted that EM is not suitable for high precision applications. As can be seen in figure 5, the number of evaluations drastically increases with precision. However, during the early stage of computations, EM algorithm is highly efficient. Thus, EM is suitable for providing a good initial guess.

## 3 Model the inverse kinematics with neural network

The architecture used for solving the inverse kinematics problems is shown in Figure 6. The single layer network consists of n outputs (joint angles) and 12 inputs [**n**, **s**, **a**, **p**] which represents a location (position and orientation) of the end-effector. As mentioned earlier, the training set have been constrained to only one solution set so that the one-to-one mapping could be achieved.

For the present work, a fast and accurate learning algorithm called as Extreme learning machine

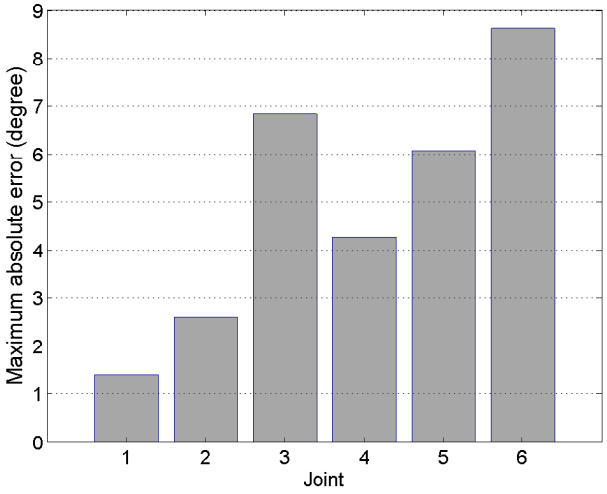


Figure 4: Maximum absolute error at each joint angle among 100 trials

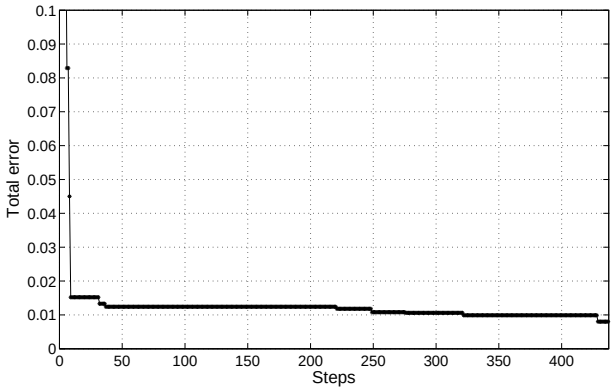


Figure 5: The performance of EM

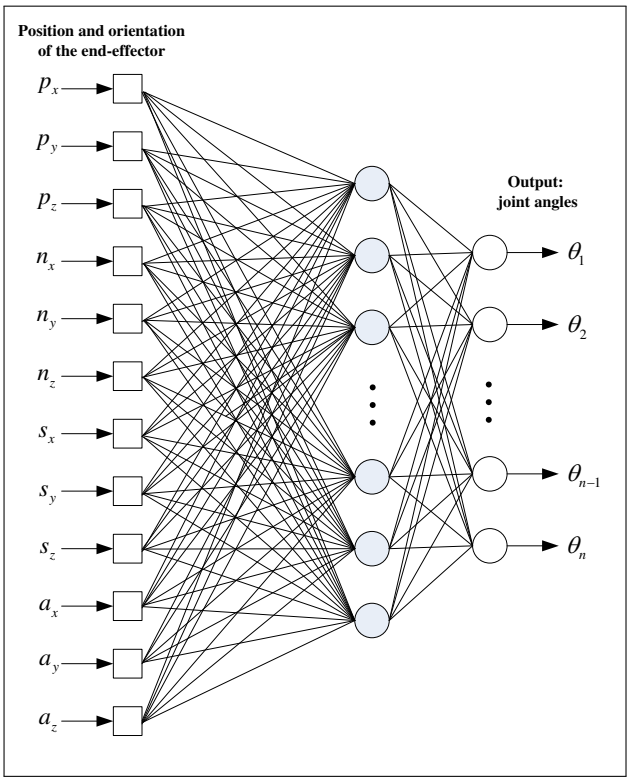


Figure 6: A general structure of the SLFN to approximate the inverse kinematics

(ELM) are used to train the neural network in modeling the inverse kinematics of robot. Test results show that the learning speed of ELM algorithm is much faster than the traditional methods. For example, the learning speed of ELM is at least 1000 and 2000 times faster than BP and SVM for solving the benchmark problem of California Housing [14]. Thus, this new training method is very suitable for solving the inverse kinematics.

### 3.1 Brief of Extreme learning machine (ELM)

ELM is a unified with randomly generated hidden nodes independent of the training data. The output of an SLFN with L hidden nodes can be represented by

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}), \quad \mathbf{x} \in R^n, \mathbf{a}_i \in R^n \tag{10}$$

Where $\mathbf{a}_i$ and $b_i$ are the learning parameters of hidden nodes and $\beta_i$ is the weight connecting in $i$th hidden node to the output node. $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the $i$th hidden nodes with respect to the input $\mathbf{x}$. Additive and RBF hidden nodes are used often in applications.

For additive hidden node with the activation function $g(x)$ (e.g. sigmoid,threshold,sin,etc.), $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \qquad b_i \in R \tag{11}$$

Where $\mathbf{a}_i$ is the weight vector connecting the input layer to the $i$th hidden node and $b_i$ is the bias of the $i$th hidden node. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors.

For RBF hidden node with activation function $g(x)$ (e.g. Gaussian), $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \parallel \mathbf{x} - \mathbf{a}_i \parallel) \qquad b_i \in R^+ \tag{12}$$

Where $\mathbf{a}_i$ and $b_i$ are the center and impact factor of $i$th RBF node. $R^+$ indicates the set of all positive real values. The RBF network is a special case of SLFN with RBF nodes in its hidden layer. Each RBF node has its own centroid and impact factor, and its output is given by a radially symmetric function of the distance between the input and the center.

For a given set of trainning samples$(\mathbf{x}_i, \mathbf{t}_i)_{i=1}^N \subset R^n \times R^m$,if the outputs of the network are equal to the targets, we have

$$f_L(\mathbf{x}_j) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_i) = t_j \qquad j = 1, 2, \cdots, N. \tag{13}$$

Above equation can be written compactly as

$$\mathbf{H}\beta = \mathbf{T} \tag{14}$$

Where

$$\mathbf{H} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1), \cdots, G(a_L, b_L, \mathbf{x}_1) \\ \vdots, \cdots, \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N), \cdots, G(a_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L} \tag{15}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad \text{and} \quad T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m} \tag{16}$$

$\beta^T$ is the transpose of a matrix or vector $\beta$. $\mathbf{H}$ is called the hidden layer output matrix of the network; the $i$th column of $\mathbf{H}$ is the $i$th hidden node's output vector with respect to input and the $j$th row of $\mathbf{H}$ is the output vector of the hidden layer with respect to input $x_j$.

Usually, when the number of training data is larger than the number of hidden nodes $N > L$, one can not expect an exact solution of the system (14).After the hidden nodes are randomly generated and given the training data, the hidden-layer output matrix $\mathbf{H}$ is known and need not be tuned. Thus, training SLFNs simply amounts to getting the solution of a linear system (14) of output weights $\beta$. Under the constraint of minimum norm least square, i.e., min $\|\beta\|$ and$\|\mathbf{H}\beta - \mathbf{T}\|$ , a simple representation of the solution of the system (14) is given explicitly as

$$\widehat{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{17}$$

Where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of the hidden-layer output matrix $\mathbf{H}$. The simple learning algorithm can be summarized as follows:

**Algorithm ELM:** Given a training set $\Re = (\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N$, activation function $g(x)$,and hidden neuron number $\overline{N}$

Step 1: Assign arbitrary input weight $\mathbf{w}_i$ and bias $b_i, i = 1, \cdots, \overline{N}$;

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$;

Step 3: Calculate the output wight $\beta : \beta = H^{\dagger}\mathbf{T}$

Where $\mathbf{H}$,$\beta$ and $\mathbf{T}$ are defined as formula (15) and (16).

# 4 Performance evaluation and discussion

**Example 1**: This simple example demonstrates that the neural network trained by the constrained data can produce a better approximation of the inverse kinematics function. An RBFN is used to approximate the inverse kinematics function of two-link manipulator. It consists of two revolute joints and two links that have the same length of 30mm. Two coordinate values x, y describe the position of the tip of the manipulator. The forward kinematics is

$$\begin{cases} x = l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2) \\ y = l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2) \end{cases} \tag{18}$$

The inverse kinematics can be described by

$$\theta_2 = Atan2(\pm\sqrt{1 - (\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2})^2}, \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}) \tag{19}$$

$$\theta_1 = Atan2(y, x) - Atan2(l_2 sin\theta_2, l_1 + l_2 cos\theta_2) \tag{20}$$

Given a desired configuration of the end-effector, there are usually two desired true solutions which correspond with the lower-elbow structure and the upper-elbow structure respectively. We test the neural network with three different cases.

1. The training set randomly sample from the whole joint space.

2. The training set randomly sample from the constrained joint space which only consisted of the positive solution (+sign in (19)).

3. The training set randomly sample from the sub joint space.

All the simulation are carried out in Matlab 6.5 environment running in an Intel(R) Core(TM) 2 Duo CPU 3.00GHz Pc. The training process of the neural network can be executed using Matlab code "newrb". The root mean squared (RMS) error goal is defined by 0.001, and the number of training sets is 1000.

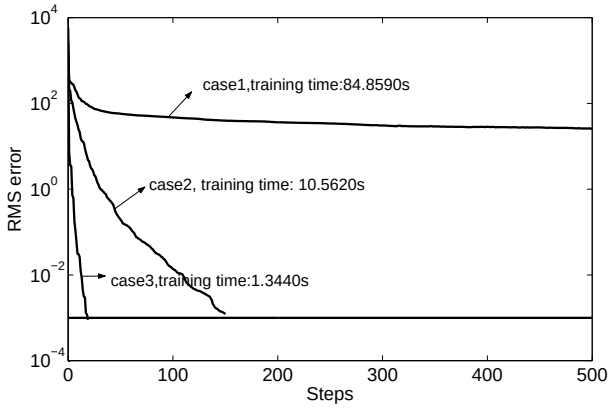


Figure 7: The training convergence performance

As can be seen in the figure 7, the same network trained by the constrained training data produces a better convergence performance.Moreover, the neural network trained within the sub joint space produces the best performance. For case 1, the training sets contain the many-to-one mapping from the joint space to the Cartesian space. It may be one reason that leads to training failure.

**Example 2**: In this example, the performance comparison of the new proposed ELM algorithm and the gradient-based learning algorithm has been conducted for an inverse kinematics of PUMA robot. The desired configuration of the end-effector is the same with section 2.3.

**Test 1:Training the network with traditional algorithm**

First, the approximate solution is calculated by EM. Set the $d\theta = 30°$, then one of the sub space is determined for each joint, as shown in table 2.

Table 2 One group of joint subspace

| Joint Number | joint 1 | joint 2 | joint 3 | joint 4 | joint 5 | joint 6 |
|---|---|---|---|---|---|---|
| Subrange (degree) | [-20.1, 39.9] | [-8.7, 51.3] | [2.1, 62.1] | [2.9, 62.9] | [11.9, 71.9] | [37.6, 97.6] |

Next, different training size, which is 100, 500 and 1000 respectively, sample randomly from the sub joint space. Other 500 data set is used for testing the performance of neural network. The root mean squared error goal and the maximum number of neurons are set as $10^{-8}$ and 500, respectively. And the spreads in three cases are experimentally selected as 3, 1.2 and 1.2 so that the RBFN can produce an appropriate performance. The training steps are repeated until the network's root mean squared error falls below goal or the maximum number of neurons are reached. Figure 8 shows the training convergence performance obtained by using different training size. The training time increases greatly with the number of the training data, as can be seen in the figure 8. Although the training error fail to reach the goal $10^{-8}$ using 500 and 1000 training data, all three trained networks are considered to achieve a good approximate performance. Since the training error successfully reach the $10^{-4}$ in three case, which is an accepted result for inverse kinematics. These conclusions can also be confirmed by the following results. Figure 9, 10 and 11 shows the testing root mean square (RMS) error at each joint angle using the corresponding networks trained above. It can be seen that the RMS error is very small. In addition, the network trained using 500 and 1000 size performs similarly. And the generalizations of both of them are better than the network trained using 100 data size. This occurs because less training data reduces the generalization of the network. However, taking into account training time, the network trained with less data size appears to be a better choice.

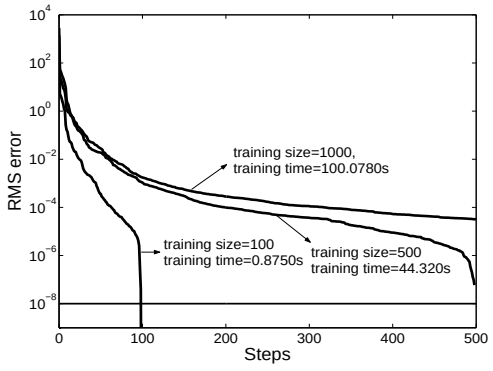


Figure 8: Network training convergence with different training size.

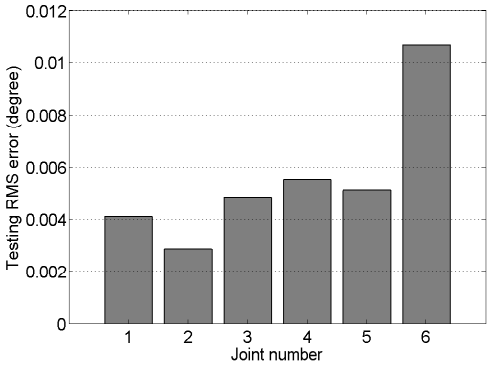


Figure 9: Absolute error at each joint angle using 100 training set

**Test 2: Training the network with ELM**

In this example, a single feedforward network with sigmoidal additive activation function is used. For ELM, the input weights and biases are randomly chosen from the range [-1, 1].To compare the results of ELM and gradient-based learning algorithm in test 1, two groups of tests use the same training/testing sets. Figure 12 shows the training RMS errors with different hidden nodes number in three cases. The corresponding testing RMS errors are plot in Figure 13. The average training time is 0.0014 s, 0.0056 s and 0.012 s, respectively. As observed from Figure 12 and 13, in general, the network trained using three

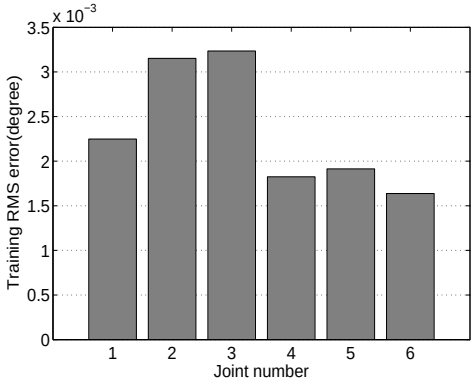


Figure 10: Absolute error at each joint angle using 500 training set.

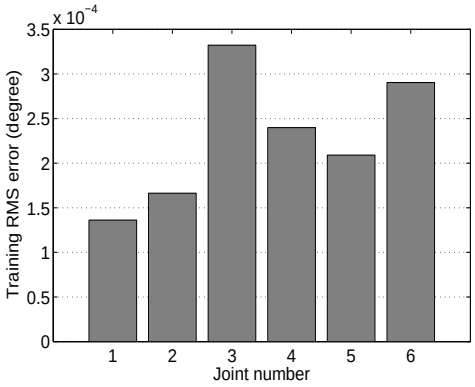


Figure 11: Absolute error at each joint angle using 1000 training set

groups of training data performs similarly. Furthermore, the lowest validation error is achieved when the numbers of hidden nodes are within the ranges [15, 50]. The results show that the generalization performance obtained by the ELM algorithm is very close to the generalization performance of gradient-based learning algorithm. However, the ELM algorithm can be simply conducted and runs much faster. According to our results, the average learning speed of ELM algorithm is at least 1000 times than the gradient-based learning algorithm.

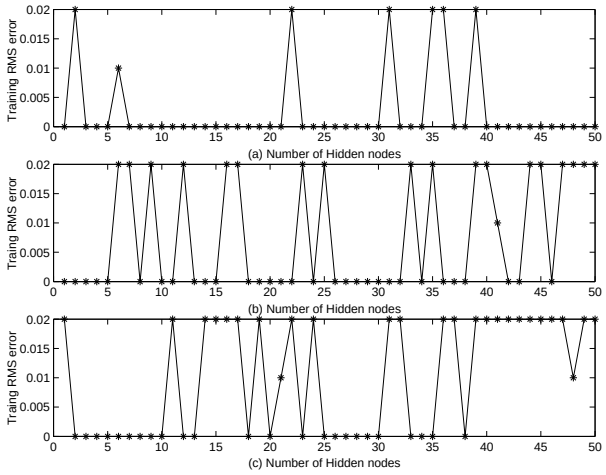


Figure 12: The training RMS error (degree) with ELM, the training size: (a) 100, (b) 500 and (c) 1000.

**Example 3**: This example demonstrates that the proposed method can be used for continuous joint space trajectory planning. The robot structure for this example is still based on PUMA 560 robot. The desired trajectory of the end-effector is a circle centered at (0.2, 0.05, 0.5) (m) with respect to the base coordinate frame and a radius equal to 0.2(m). The trajectory is discretized into 72 equally spaced points. To ensure the existence of solution, the joint limitations are released in this example. Moreover, noting that multiple solutions do exit, in order to prevent a sudden jump to another solution, a unique orientation is assigned and the approximate solution for each of the successive points is given by the solution of the preceding point. For example, if the calculated solution of the $k$ point is denoted by $\theta_k$, the joint variable limitations are set as $[\theta_k - d\theta, \theta_k + d\theta]$ for the $k$+1 point, instead of re-computing the approximate solution. The computed joint trajectories and the corresponding total error (sum of the position and orientation error) are plotted in Figure 14 and Figure 15, respectively. It should be noted

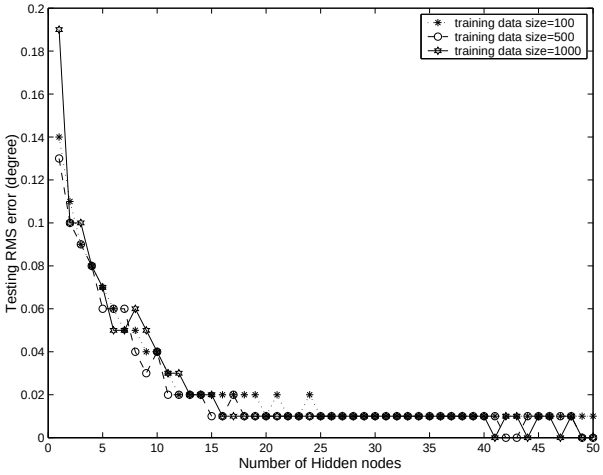


Figure 13: The testing RMS error with ELM.

that the trajectory in Figure 14 is just one of the multi-trajectory for PUMA robot.

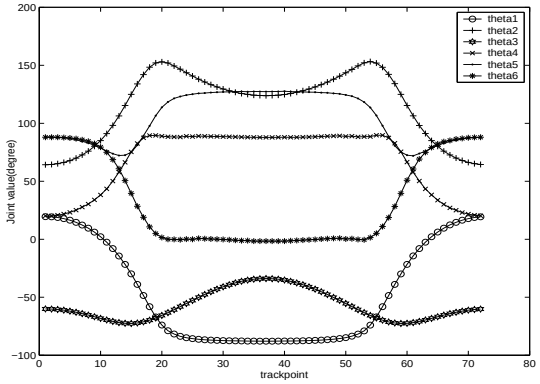


Figure 14: Computed trajectories of the joints.

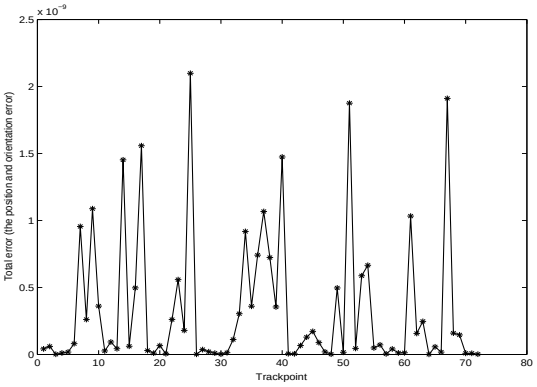


Figure 15: Total error at each track point

The results in Figure 15 show that the idea of using a neural network has produced an excellent approximation of the inverse kinematics function. Although neural network solutions are usually not suited for high precision robotic application, high precision results are achieved here. This occurs because the joint varies are limited within a small space when training network.

# 5 Conclusions

The proposed hybrid approach combined the electromagnetism-like method and the neural network to solve the inverse kinematics problem. Unlike the traditional neural network approaches that generate the training data from the whole joint space, the neural network in the proposed approach collects the training data from a sub joint space, in which the training set is constrained to only one solution set so that the one-to-one mapping is achieved. Another important feature of the proposed approach is to use an efficient learning algorithm, ELM, to train the neural network. The learning speed of this novel training algorithm can be thousands of times faster than traditional feedforward network learning algorithms while obtaining better generalization performance. The results show that the proposed hybrid approach has not only greatly reduced the computation time but also improved the precision.

## Bibliography

[1] Bruno Siciliano, Oussama Khatib, *Springer Handbook of robotics*,Springer Press,2008.

[2] H JACK, DMA LEE, RO BUCHAL and WH ELMARAGHY, Neural networks and the inverse kinematics problem, *Journal of intelligent manufacturing*,4:43-66,2003.

[3] FL Lewis,Neural network control of robot manipulators,*IEEE Expert*,11(3):64-75,1996.

[4] BB Choi and C Lawrence,Inverse kinematics problem in robotics using neural networks,NASA Technical Memorandum-105869.

[5] Z Binggul, HM Ertunc and C Oysu, Comparison of inverse kinematics solutions using neural network for 6R robot manipulator with offset,*In Proceedings of the 2005 Congress on Computational Intelligence Method and Application*,pp:1-5.

[6] AS Morris , A Mansor,Finding the inverse kinematics of manipulator arm using artificial neural network with look-up table. *Robotica*,15:617-625,1997.

[7] JA Driscoll,Comparison of neural network architectures for the modeling of robot inverse kinematics,*In Proceedings of the 2000 IEEE*,3:44-51,2000.

[8] SS Yang, M Moghavvemi and John D Tolman,Modelling of robot inverse kinematics using two ANN paradigms, *In Proceedings of TENCON2000 Intelligent System and Technologies for the New Millennium*,3:173-177,2000.

[9] Shital S, Chiddarwar N and Ramesh Babu,Comparison of RBF and MLP neural networks to solve inverse kinematic problem for 6R serial robot by a fusion approach,*Engineering Applications of Artificial Intelligence,*, 23(7):1083-1092,2010.

[10] PY Zhang, TS Lu and LB Song, RBF neworks-based inverse kinematics of 6R manipulator,*Int. Journal of advanced manufacturing technology*,26:144-147,2004.

[11] Eimei Oyama, Arvin Agah and Karl F, A modular neural architecture for inverse kinematics model learning,*Neurocomputing*, 38(40):797-805,2001.

[12] Srinivasan Alavandar, MJ Nigam, Neuro-Fuzzy based approach for inverse kinematics solution of industrial robot manipulators, *Int. J. of computers, Communication and Control*, 3(3):224-234,2008.

[13] Karlra P, Prakash NR, A neuro-genetic algorithm approach for solving inverse kinematics of robotic manipulators, *IEEE International Conference on Systems, Man and Cybernetics*,2:1979-1984,2003.

[14] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew, Extreme learning machine: A new learning scheme of feedforward neural networks, *In Proceedings of IEEE International joint conference on Neural Networks*,2:985-990,2004.

[15] Guang-Bin Huang, Lei Chen, Enhanced random search based incremental extreme learning machine,*Neurocomputing*, 71(16-18):3460-3468,2008.

[16] Birbil SI, Fang SC, An electromagnetism-like mechanism for global optimization,*Journal of Global Optimization*, 23(3):263-282,2003.

[17] Birbil SI, Fang SC, Sheu RL, On the convergence of a population-based global optimization algorithm,*Journal of global optimization*, 30:301-318,2004.

[18] Wang LCT, Chen CC, A combined optimization method for solving the inverse kinematics problem of mechanical manipulators,*IEEE Transaction on Robotics and Automation*, 7(4):489-499,1991.