Back Propagation Through Links: A New Approach to Kinematic Control of Serial Manipulators

Ran Gazit*
Gravity Probe B, Hansen Labs
Stanford University
Stanford, CA 94305-4085

Bernard Widrow†
Information Systems Laboratory
Stanford University
Stanford, CA 94305-4055

Abstract

We present a new approach to neural control of serial manipulators, based on the sequential nature of the forward kinematics equations. A neural network is trained to compute the angle between two adjacent links, using the location error of the connecting joint as an input. This angle is then used to derive the location of the next joint, according to a single link kinematic equation. The procedure is repeated until all the links angles are computed. When embedded in a closed loop controller, this algorithm provides smooth operation of a serial manipulator with any number of links.

The neural network is trained by backpropagating the end–effector location error through the links equations, in a similar way to Back Propagation Through Time. The training procedure does not involve known solutions of the inverse kinematics problem. Moreover, no retraining of the network is required when adding or removing links. Several examples demonstrate the manipulator peformance for three, four and six link robot arms.

1 Introduction

In a serial manipulator, the first joint is usually fixed, and the last joint is free and acts as the manipulator's end–effector. To move the end–effector to a desired point or along a desired path, the robot controller is required to solve the Inverse Kinematics Problem. This problem involves the computation of a sequence of links angles that will position the end–effector at the desired location.

The computational complexity involved with the numerical solution of the Inverse Kinematics Problem, and the capability of neural networks to approximate arbitrary functions, attracted many researchers to apply neural networks to this problem [1-8]. Most of these works use known solutions of the inverse problem to generate input–output patterns for the network training process. In all cases, the network computes all the angles between the links in one step, using the desired end–effector location as the network input.

These solutions disregard the sequential nature of the problem, and overlook the striking resemblance between the forward kinematics of a serial manipulator and the dynamics of a discrete time system. In this work we make use of this quality, and train a neural controller of a serial manipulator by using a method similar to Back Propagation Through Time [9].

In the next section we present some background material. A "one step" solution to the Inverse Kinematics Problem demonstrates how the Jacobian matrix of the forward kinematics can be used to backpropagate the location error of the end–effector. This eliminates the need for known solutions in the network training process. The use of the Jacobian matrix is further extended in section 3, where we present a sequential approach to the solution of the Inverse Kinematics Problem. Section 4 concludes the paper with suggestions for future research.

2 Background

2.1 Problem Statement

Consider the planar robot arm depicted in figure 1. We assume that all the links have constant and equal length, set for simplicity to unity length. The end–effector location $[x_e, y_e]$ is given by:

$$x_e = \cos\theta_1 + \cos(\theta_1 + \theta_2) + \cdots \quad (1)$$

$$y_e = \sin\theta_1 + \sin(\theta_1 + \theta_2) + \cdots \quad (2)$$

where θ_k is the angle between link $k-1$ and link k, $k = 1, 2 \ldots L$, and L is the number of links. These

*Ph.D. Candidate, Dept. of Aeronautics & Astronautics.
†Professor, Dept. of Electrical Engineering

0-7803-2722-5/95 $4.00 © 1995 IEEE

equations represent the forward kinematics of the manipulator. The Inverse Kinematics Problem is defined as follows: Find a sequence of angles $\{\theta_1, \theta_2, \ldots \theta_L\}$ that will position the end–effector at a desired point $[x_d, y_d]$.

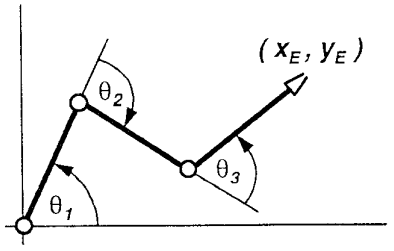

Figure 1: A planar serial manipulator

2.2 "One Step" Solution

The common approach to the solution of this problem by using a neural network [1-8], is to calculate all the links angles in one step. The input to the neural network is the desired end–effector location (see figure 2). The network is usually trained by presenting it with input–output patterns, which are generated by analytical solutions of the inverse problem.

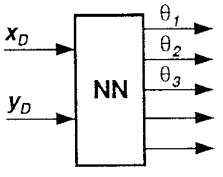

Figure 2: Neural network configuration in a "one step" solution of the Inverse Kinematics Problem

However, the network can be trained without relying on known solutions, in the following way: a desired end–effector location $[x_d, y_d]$ is introduced to the network. The network calculates the links angles $\{\theta_1, \theta_2, \ldots \theta_L\}$, and those are substituted into the forward kinematics equations (1,2) to yield the actual end–effector location $[x_e, y_e]$. The location error

$$\mathbf{e} \triangleq [x_d - y_e, y_d - y_e]^{\mathrm{T}} \tag{3}$$

is then multiplied by the Jacobian matrix

$$\frac{dF}{d\theta} = \begin{bmatrix} -\sin\theta_1 - \sin(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & \ldots \\ \cos\theta_1 + \cos(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & \ldots \end{bmatrix} \tag{4}$$

The product is used to compute the required change in the weight matrices, according to the LMS algorithm [10]:

$$\Delta W = -\mu \frac{\partial \mathbf{e}^{\mathrm{T}} \mathbf{e}}{\partial W} = 2\mu \left[\frac{dF}{d\theta}\right]^{\mathrm{T}} \mathbf{e} \tag{5}$$

This training procedure is described in figure 3. In this figure NN refers to the neural network, and **F** refers to the forward kinematics equations (1,2).

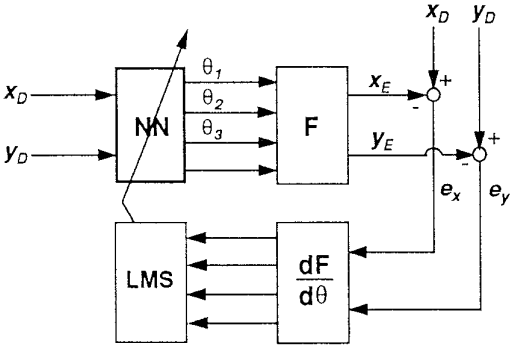

Figure 3: Training a "one step" controller, using the Jacobian matrix of the forward kinematics equations

Note that any configuration change of the manipulator (such as adding or removing links) will require an appropriate change in the network structure (adding or removing output nodes). In that case, the training procedure should be repeated, using the relevant Jacobian matrix.

2.3 Simulation Results

The performance of the "one step" controller is demonstrated by the following example of a 3-link robot arm. A neural network with 2 inputs (x_d, y_d) and 3 outputs $(\theta_1, \theta_2, \theta_3)$ was trained according to the procedure described above. The training input was a set of desired end–effector locations, equally spaced in the reachable domain of the robot arm.

Figure 4 shows the controller performance after convergence of the network weights. In this example the robot base is fixed at the origin of the XY plane, and the end–effector is commanded to follow a set of straight lines. It seems that the neural network succeeded to learn the inverse kinematics of the manipulator, and the end–effector does follow the straight lines as commanded.

However, when looking at the robot arm itself (figure 5), we observe an unacceptable behavior. At the same time that the end–effector is smoothly tracking

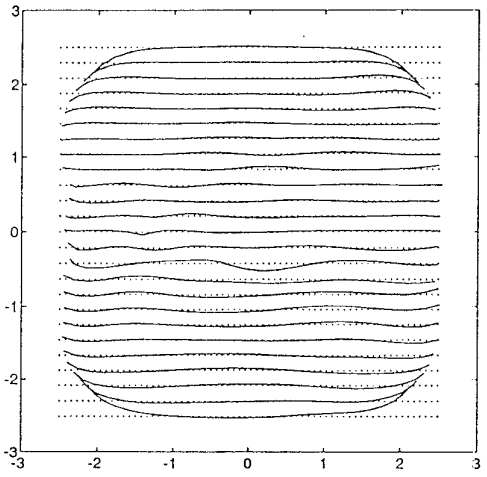

Figure 4: End–effector trajectories (solid lines) of a 3-link robot arm, commanded to follow straight lines (dotted)

a straight line, the angles between the links change abruptly. Since the neural network was trained to provide a static mapping between points in the XY plane and a sequence of links angles, it does not necessarily guarantee smooth change of the angles when the end–effector moves between two close points. The controller can therefore switch back and forth between redundant solutions of the inverse kinematics problem.

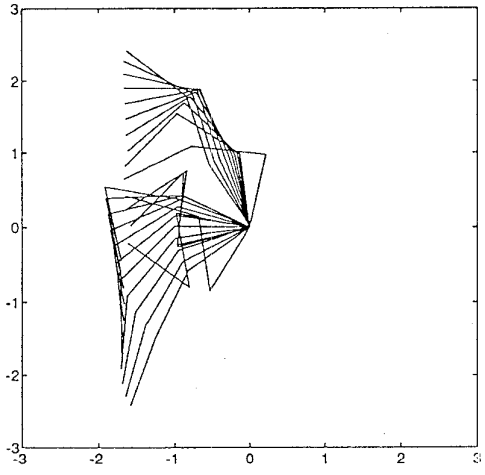

Figure 5: 3-link robot arm following a straight line

The next section describes a new approach to the neural control of serial manipulators, which aims to solve these difficulties.

3 Back Propagation Through Links

3.1 Global Solution

By defining the links angles relatively to a common base line (see figure 6), we can describe the forward kinematics of a planar serial manipulator with L links as follows:

$$x_{k+1} = x_k + \cos\theta_k \quad (6)$$

$$y_{k+1} = y_k + \sin\theta_k \quad (7)$$

where: $k = 1, 2 \ldots L$ is the joint index, $[x_k, y_k]$ describe the location of joint k, and θ_k is the angle between link k and the x axis.

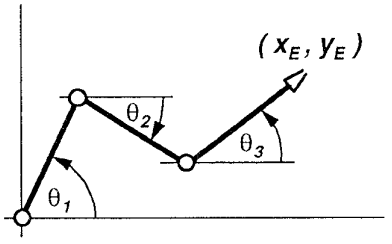

Figure 6: A planar serial manipulator with angles defined relatively to a base line

The set of non linear equations (6,7) is in a form similar to the state equations of a dynamic, discrete time system:

$$\mathbf{z}_{k+1} = F(\mathbf{z}_k, u_k) \quad (8)$$

where the state vector is $\mathbf{z} \equiv [x, y]^{\mathrm{T}}$ and the control variable is $u \equiv \theta$. We wish to find the control sequence $\{\theta_1, \theta_2, \ldots \theta_L\}$ that will bring the end–effector $[x_{L+1}, y_{L+1}]$ to a desired point $[x_d, y_d]$. This is a terminal control problem, where the cost function to be minimized is the error in the end–effector position:

$$J = \mathbf{e}^{\mathrm{T}}\mathbf{e} \quad (9)$$

and

$$\mathbf{e} \triangleq [x_d - x_{L+1}, y_d - y_{L+1}]^{\mathrm{T}} \quad (10)$$

Nguyen and Widrow [11] and others [9, 12] have used an algorithm called *Back Propagation Through Time* (BPTT) to solve similar terminal control problems. According to this algorithm, a neural net is fed with $\mathbf{z}_k$, the state vector at index k. The network then computes the control u_k, required to move the state from index k to the next index $k + 1$. Usually, this index refers to time. In our case, the index k refers to links

along the robot arm. It is therefore appropriate to denote the following method of robot arm control as *Back Propagation Through Links* (BPTL).

The basic element in the controller is a neural network (see figure 7) which calculates the required orientation of link k, based on the location of joint k. The angle computed by the neural network and the joint location are used together to calculate the location of the next joint, by using equations (6,7). This procedure is repeated until all the angles are found.

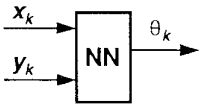

Figure 7: The neural network configuration in Back Propagation Through Links

Since the first joint is located at the origin, the procedure is initialized by translating the robot arm to minus the desired point, and feeding this value as the first joint location. The controller then solves for the angles that will bring the end–effector to the origin. These angles are equal to the angles required to bring the end–effector to the desired point, when the first joint is fixed at the origin.

The complete controller structure is depicted in figure 8. In this figure $\mathbf{F}_1$ refers to the equations describing the forward kinematics of one link (6,7).

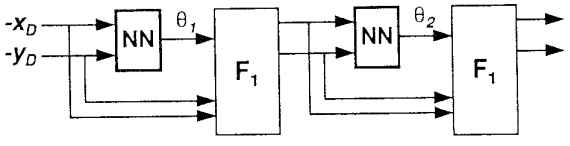

Figure 8: Back Propagation Through Links - controller structure

The neural network is trained by back propagating the location error of the end–effector, through all the links. As was shown in the previous section, we use the Jacobian matrix of one link

$$\frac{dF}{d\theta_k} = \begin{bmatrix} -\sin\theta_k \\ \cos\theta_k \end{bmatrix} \tag{11}$$

to back propagate the location error through the link equation at index k. By multiplying the location error by the Jacobian matrix, we get the error gradient required to calculate the change in the weight matrix. The errors that were back propagated through the neural network, are summed with the location errors of joint k, to produce the location error at joint $k-1$. This is the standard Back Propagation Through Time approach to terminal control, applied here to back propagating the error through the links.

This approach eliminates the need for retraining the neural network after a configuration change, since the network is the same for all links. However, it does not solve the problem of redundant solutions to the inverse kinematics problem. In order to do that, we have to provide the controller with information on the current geometry of the manipulator. By doing so, we can look at the motion of the end–effector between two close points, and solve for the required *change* in the geometry, instead of solving for the geometry itself.

3.2 Incremental Solution

Consider a small change in θ_k, denoted as $\Delta\theta_k$. This will cause an appropriate change in x_{k+1} and y_{k+1}. After applying small changes to all the links angles, the robot arm geometry is described by:

$$x_{k+1} + \Delta x_{k+1} = x_k + \Delta x_k + \cos(\theta_k + \Delta\theta_k) \tag{12}$$
$$y_{k+1} + \Delta y_{k+1} = y_k + \Delta y_k + \sin(\theta_k + \Delta\theta_k) \tag{13}$$

Subtract the original one link equations (6,7) and get:

$$\Delta x_{k+1} = \Delta x_k + \cos(\theta_k + \Delta\theta_k) - \cos\theta_k \tag{14}$$
$$\Delta y_{k+1} = \Delta y_k + \sin(\theta_k + \Delta\theta_k) - \sin\theta_k \tag{15}$$

We wish to solve for the sequence of angle increments $\{\Delta\theta_1, \Delta\theta_2, \ldots \Delta\theta_L\}$ that will change the end–effector location from $[x_{L+1}, y_{L+1}]$ to $[x_{L+1} + \Delta x_{L+1}, y_{L+1} + \Delta y_{L+1}]$. This is a similar terminal control problem to the one we faced in the previous section. We use a similar controller architecture (see figure 9), where an additional parameter – the current value of the link angle – is fed to each neural network and link equations. The term $\mathbf{F}_\Delta$ in figure 9 refers to the difference equations that describe the local kinematics of a single link (14,15).

In order to initialize the procedure at a value different then zero, we set the initial location increment to

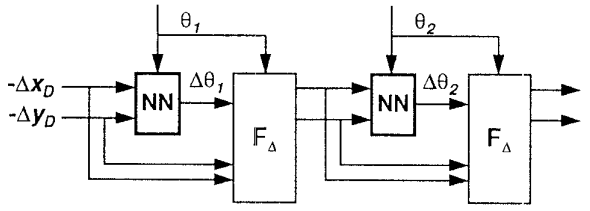

Figure 9: Incremental Back Propagation Through Links - neural network and link equations sequence

minus the desired change at the end–effector location:

$$\Delta x_o = -\Delta x_d \quad (16)$$

$$\Delta y_o = -\Delta y_d \quad (17)$$

In that case, the output of the final block $[\Delta x_{L+1}, \Delta y_{L+1}]$ should be zero.

The natural way to obtain Δx_d and Δy_d is to put the controller inside a closed loop, as described in figure 10. In this figure, **B** denotes a block which contains a sequence of a neural network and a single link difference equations, such as described in figure 9. For simplicity, we refer in this figure to the x coordinate only. Note that the building blocks of this controller are identical for all links.

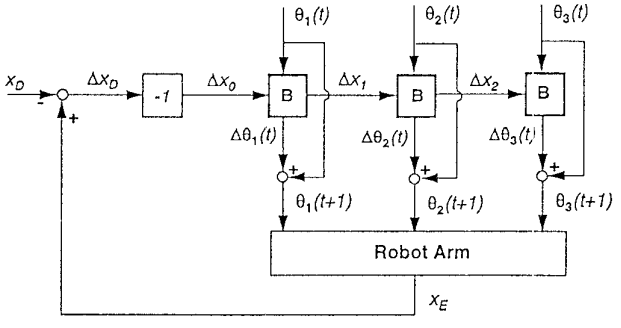

Figure 10: Incremental Back Propagation Through Links - closed loop controller structure

3.3 Simulation Results

The next figures show a serial manipulator following straight lines. The manipulator was controlled according to the closed loop scheme described above. The neural network element of the closed loop controller was initially trained by using a 3-link configuration (figure 11). The same network, with no retraining was used to control a 4-link manipulator (figure 12) and a 6-link manipulator (figure 13). The only difference in the controller structure is additional NN and $\mathbf{F}_\Delta$ blocks.

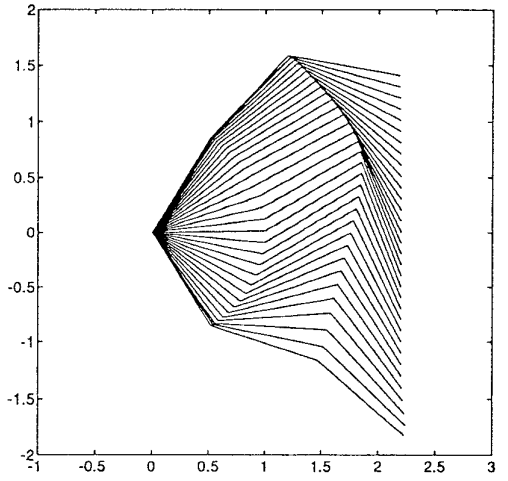

Figure 11: Incremental Back Propagation Through Links - 3-link robot arm

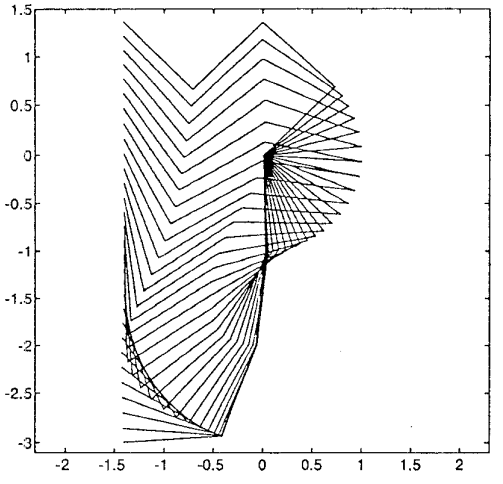

Figure 12: Incremental Back Propagation Through Links - 4-link robot arm

4 Conclusions

A new approach to kinematic control of serial manipulators was introduced. This approach is based on the sequential nature of the forward kinematics equations, and uses the same neural network repeatedly to compute only one link angle at a time. It is similar in nature to Back Propagation Through Time, where the time index there is replaced by a joint index here.

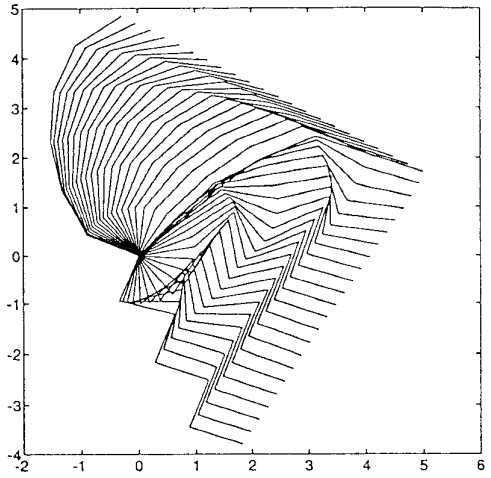

Figure 13: Incremental Back Propagation Through Links - 6-link robot arm

The training procedure of the neural network does not require known solutions of the inverse kinematics problem, and need not be repeated when adding or removing links.

Future work will extend the approach presented here to solve the general inverse kinematics problem, for the 4D Hartenberg-Denavit transformations. Different arm structures, such as a truss with no revolute joints could also be treated in the same way. The simple cost function used in this work should be augmented with additional terms, such as a requirement for a desired end–effector orientation, constraints on link angles or changes in angles during a unit time, and obstacles avoidance, including avoidance of self collision between far away links of the same arm.

Acknowledgments

The first author gratefully acknowledges the support of the FAA's Satellite Program Office (AGS-100) and the FAA Technical Center. Mr. Y.C. Chao and Mr. M. Moataz of Stanford University participated in early stages of this work.

References

[1] R. K. Elsley, "A learning architecture for control based on back-propagation neural networks," in *International Conference on Neural Networks*, vol. 2, pp. 587–594, IEEE, July 1988.

[2] G. Josin, D. Charney, and D. White, "Robot control using neural networks," in *International Conference on Neural Networks*, vol. 2, pp. 625–631, IEEE, July 1988.

[3] A. Guez and Z. Ahmad, "Solution to the inverse kinematics problem in robotics by neural networks," in *International Conference on Neural Networks*, vol. 2, pp. 617–624, IEEE, July 1988.

[4] S. Lee and R. M. Kil, "Robot kinematic control based on bi-directional mapping neural network," in *International Joint Conference on Neural Networks*, vol. 3, pp. 327–335, 1990.

[5] T. Yabuta and T. Yamada, "Possibility of neural networks controller for robot manipulators," in *International Conference on Robotics and Automation*, pp. 1686–1691, IEEE, May 1990.

[6] J. M. Zurada, M. Kavari, and J. H. Lilly, "Robot kinematics modeling using multilayer feedforward neural networks," in *Artificial Neural Networks in Engineering*, pp. 785–790, ASME Press, November 1992.

[7] L. C. Rabelo and X. J. R. Avula, "Hierarchical neurocontroller architecture for robotic manipulation," *IEEE Control Systems Magazine*, vol. 12, no. 2, pp. 37–41, April 1992.

[8] K. Liu and J. P. H. Steele, "A new artificial neural systems architecture and its application to robot control," in *Artificial Neural Networks in Engineering*, pp. 505–510, ASME Press, November 1993.

[9] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1590, October 1990.

[10] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Prentice-Hall, 1985.

[11] D. H. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control Systems Magazine*, pp. 18–23, April 1990.

[12] E. S. Plumer, *Optimal Terminal Control Using Feedforward Neural Networks*. PhD thesis, Stanford University, August 1993.