Proceedings of the 1993 International
Symposium on Intelligent Control
Chicago, Illinois, USA - August 1993

Endpoint Position Control Using Biological Concepts

Kelly Ann Smith & Michael G. Littman

Department of Mechanical and Aerospace Engineering
Princeton University
Princeton, NJ 08544

Abstract

A biologically inspired algorithm has been refined for position control of a multilink robotic manipulator. In this approach, changes in joint angles are computed by a method of successive forward approximation in a manner that is naturally parallel and very simple. Improvements to the algorithm provide nearly uniform convergence for all motions regardless of direction and location in the workspace. Neural networks may be used to further improve trajectory control.

1 - Introduction

A manipulator with more degrees of freedom (DOF) than necessary to perform a task is *redundant*. For example, a planar manipulator needs only two DOF to reach any point in the x-y plane. A redundant manipulator would have three or more DOF, and it may achieve a given endpoint position in an infinite number of ways.

The human body in profile can be thought of as a seven link redundant manipulator, moving in a plane (Figure 1).

Figure 1. Seven-link representation of the human body in profile.

We have developed such a manipulator in our laboratory, and we refer to it as SLIM (Skill-Learning Intelligent Manipulator). One of the goals of our work on SLIM is for movement to occur anthropomorphically. For this reason, we are interested in algorithms which form biological-like trajectories.

In most instances, there are infinitely many postures that a redundant manipulator may take to reach a given endpoint. In determining a posture, it is often beneficial to consider other factors, such as balance or obstacle avoidance, which makes some postures more desirable than others. An important advantage of redundancy is that it provides postural flexibility to satisfy multiple goals. These include the ability to reach around obstacles and avoid undesirable joint positions. Redundancy may also provide the ability to distribute loads throughout a structure, to optimize dynamic response, to minimize joint torques, or to lessen shock [1-8].

It is also well-established that there are many ways to manage link redundancy, although there is no general agreement as to a best method. The most common approach involves the Moore-Penrose pseudoinverse. We will discuss some of the drawbacks of this method and explore the use of an alternative biologically inspired approach. We will also discuss how neural networks may be used to optimize its performance.

1.1 - Inverse Kinematics of a Redundant Manipulator

The endpoint of a manipulator arm is given by the vector X, made up of m Cartesian coordinates. X is a function of Θ, a measure of n joint angles (Figure 2).

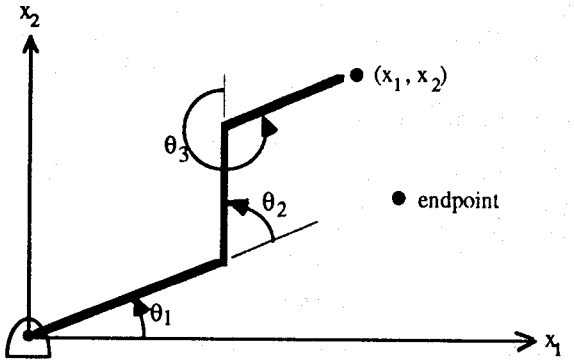


Figure 2. This figure demonstrates the relationship between angles θ_1, θ_2, θ_3 and Cartesian coordinates x_1 and x_2.

To move the end of the arm toward the desired endpoint, the appropriate joint velocities ($\dot{\Theta}$) need to be determined. Since $X = f(\Theta)$, then

$$\dot{X} = \frac{dX}{d\Theta}\dot{\Theta} \quad \text{or} \quad \dot{X} = J\dot{\Theta} \qquad (1)$$

where $J = \frac{dX}{d\Theta}$ is the Jacobian matrix of dimension $(m \times n)$. When $m = n$, $\dot{\Theta}$ may be found by taking the inverse of J, or

$$\dot{\Theta} = J^{-1}\dot{X} \qquad (2)$$

Since J is not square for redundant manipulators $(n > m)$, the Moore-Penrose right pseudoinverse or weighted right pseudoinverse of J can be used to determine $\dot{\Theta}$.

$$\dot{\Theta} = J^{R}X = J^{T}(JJ^{T})^{-1}\dot{X} \qquad (3)$$

or

$$\dot{\Theta} = J^{WR}\dot{X} = W^{-1}J^{T}(JW^{-1}J^{T})^{-1}\dot{X} \qquad (4)$$

0-7803-1206-6/93/$3.00 ©1993 IEEE

where W is a weighting matrix of dimension $(n \times n)$.

1.2 - Managing Redundancy

Our goal has been to find a simple algorithm to manage redundancy that is easy to implement and may be mapped onto a parallel computer. The algorithm must work well both inside and outside the reach of the arm, and at singularities. Also, a manipulator should begin a trajectory immediately after a desired endpoint is given; the delay, or latency, should be as small as possible. The algorithm should also be generalizable to other problems, such as systems with many links or force-position control. The belief that nature has produced a very successful redundant manipulator leads us to consider biologically inspired algorithms.

The approach we have been investigating is that of Berkinblit and his colleagues, who found that spinalized frogs use a method of successive approximation to attain a new endpoint [9]. This method involves a simple cross-product term for the angular velocity of each joint, independent of the others. The necessary movements are obtained in parallel. Besides having the advantage of inherent parallelism, the manipulator begins to move even before the algorithm has converged. This method has problems at singularities; however, the use of muscle *synergies* can resolve these difficulties. A *synergy* is a group of muscles which act together to achieve a single goal, such as the multijoint task of the withdrawal of an arm from a dangerous object. The inclusion of reflex synergies also has benefits in improving convergence of the algorithm.

Hinton also proposes a method of controlling a manipulator arm based on parallel control [10]. He suggests using a "motion blackboard" which represents the static and dynamic parameters of the arm at any instant. Joint computations are performed in parallel, and all joints have access to the previous values put on the motion blackboard. Hinton's structure may be combined along with Berkinblit's algorithm [9] to form a method of parallel successive approximation.

Other methods for managing redundancy include a variation of inverse kinematics called the Jacobian null-space method. This has been used by many to simultaneously achieve endpoint positioning and other goals, which include avoiding obstacles [2-3], singularities and unwanted joint torques [4]. Nakamura, *et al.* satisfy a number of objectives using a task priority system [5].

Use of the null space is helpful when dealing with multiple objectives; however, it is computationally expensive, since the inverse Jacobian must constantly be re-evaluated. The applications of many other algorithms are also viable, but most require many computations to achieve a result. This causes a delay before any movement occurs. Also, it is unlikely that these calculations are performed in biological movements.

Pipelined and parallel processing can be used to reduce this time. Pipelined processing works well if many serial calculations are needed, such as finding the inverse of a matrix. Pipelining an inverse kinematics system has the capacity to improve computation time; however, there is a delay before the first set of calculations is finished where there is no movement.

There is generally a much smaller time delay for calculations computed in parallel over those which are pipelined. Zhang and Paul [11] take advantage of this in a parallel method of performing the inverse kinematics task. Movement begins sooner in a parallel system, and accuracy is not necessarily sacrificed. It is simply a matter of performing a few more iterations with the parallel method to achieve the accuracy obtained by a pipelined method. Since our algorithm is inherently parallel, we enjoy the advantages of quick processing and no delay in movement. Both of these qualities contribute to biological-like movements.

2 - Berkinblit-Hinton Algorithm

The biologically inspired Berkinblit-Hinton algorithm is quite simple. A basic cross-product term provides joint velocities in the form of angle increments:

$$\Delta\theta_i = k_i |r_i||e| \sin(\phi_i) \quad \text{or} \quad \omega_i * \Delta t = k_i\, r_i \times e \qquad (5)$$

These increments represent velocities taken over a constant change in time, Δt. A scalar gain, k_i, multiplies the cross-product of r_i and e. r_i is the vector from joint i to the end of the arm. e is the error vector, which is measured from the actual endpoint of the arm to the desired endpoint. θ_i is the angle at joint i, as defined in Figure 2. ϕ_i is the angle between r_i and the error vector (Figure 3).

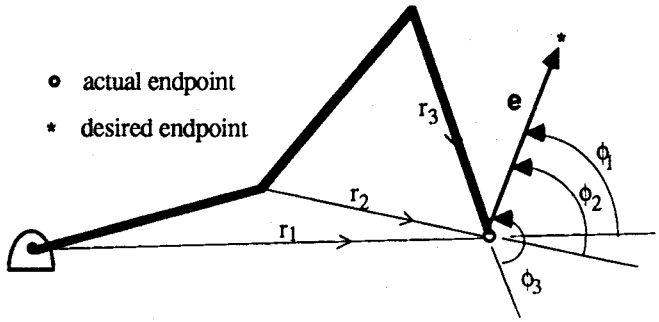


Figure 3. This figure defines r, e and ϕ.

$\Delta\theta_i$ is computed for each of the i joints simultaneously, and all links are commanded to move at each iteration.

2.1 - Performance of Algorithm

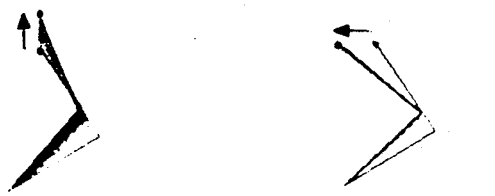
This paper discusses only those results found in simulation where we assumed there was no gravity (no dynamics). This algorithm was tested on an n-link arm, with the constraints that it could only move into postures a human arm could obtain. In this section, we discuss the results of a two-link case for simplicity. Similar results on the corresponding three-link arm are discussed in *Section 3 - Application to N-Link Systems*.

A measure of the algorithm's performance is obtained by moving the desired endpoint one-sixth of the total arm length as a step input. The number of iterations needed for convergence to within one-tenth of the original error is determined. An example data point is shown in Figure 4. The sum of all iterations in the four planar directions is found for 33 points in the first quadrant. This measure of performance is defined as I_A.

Performance near the origin is also a concern. Using the same method as before, the number of iterations until convergence is found when the endpoint is near the origin. The error vector is one-sixtieth of the total arm length. Two values are summed: a horizontal error and a vertical error. This second measure is I_B.

As shown in these figures, the endpoint trajectories are quite smooth, with minimal instabilities. It is also demonstrated that this method produced large changes in endpoint initially; the changes become smaller and smaller as the endpoint nears its goal. This response is very desirable because of its similarity to the behavior of humans and other

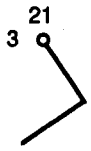
animals. It is also desirable because of other properties. Convergence may require a few more iterations than an inverse kinematics method. However, those iterations consist only of three very simple calculations for each link, unlike the inverse Jacobian which requires work of order n^3 [12].



$\theta_1 = 46.88°$ x1 = 0.100
$\theta_2 = 63.61°$ x2 = 0.500
iterations: 21

$\theta_1 = 41.81°$ x1 = 0.000
$\theta_2 = 96.38°$ x2 = 0.400
iterations: 3

Figure 4a. The two simulations above are shown in the notation below. For a gain of $k_i = 4$, it took 21 iterations to move the actual endpoint from (.1, .4) to (.1, .5), and 3 iterations to move from (.1, .4) to (0, .4). (Each link is .3 long.)

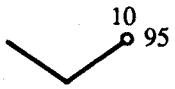


$(x_1, x_2) = (.1, .4)$

Figure 4b. The performance in Figure 4a is symbolized by this diagram.

2.2 - *Drawbacks to Method*

In situations where an extension or retraction of the arm is needed, many iterations are needed. If the arm simply needs to be rotated, it only takes a few iterations (see Figure 5).



$(x_1, x_2) = (.5, 0)$

Figure 5. Using the notation from Figure 4, iterations for motion upwards and to the right are noted.

The slow convergence when retracting and extending is due to the sine term in the cross-product. The angle ϕ_1 between r_1 and the error vector is 0 when extending and π when retracting. Since sin (0) = sin (π) = 0, θ_1 does not change. Since θ_2 must change before θ_1 changes, this process becomes very slow and many iterations are needed for convergence. If sin (ϕ_2) = 0 also, the arm does not move at all. I_A = 4800 iterations, and most of the iterations are found along extending/retracting lines.

There is another region where convergence is slow. Near the origin, r_1 is very small, which causes $\Delta\theta_i$ to be small. The arm moves very slowly, as shown by the large iteration count in Figure 6 (I_B = 3372 iterations).

$(x_1, x_2) = (.01, 0)$

Figure 6. Using the notation from Figure 4, iterations for motion upwards and to the right are noted.

2.3 - *Solutions to Problems*

The basic Berkinblit-Hinton algorithm has two main drawbacks: slow convergence when retracting/extending and slow convergence near the origin. The first problem can be solved by looking at biology. Humans have reflexes in all of their joints; these reflexes serve to synchronize the movements of many muscles to achieve a result. This coordinated movement is a synergy. There are, of course, many reflexes involved with extending and retracting the limbs. For this reason, a *reflex* term is now added which helps to collapse and straighten the arm. This is based on the error between the actual length of the arm and the desired length. The cross-product term for the first link is as follows:

$$\omega_{ci} * \Delta t = k_{ci}\, r_i \times e \quad \text{or} \quad \Delta\theta_{ci} = k_{ci} |r_i| |e| \sin(\phi_i) \qquad (6)$$

where k_{ci} is the cross-product gain of link i. The reflex term is the corresponding dot product:

$$\omega_{ri} * \Delta t = k_{ri}\, r_i \bullet e \quad \text{or} \quad \Delta\theta_{ri} = k_{ri} |r_i| |e| \cos(\phi_1) \qquad (7)$$

where k_{ri} is the reflex gain of link i. In a multilink arm, all the angles needed to collapse the arm are dependent on $\Delta\theta_1$. The reflex gain k_{ri} is as follows: $k_{r2} = -2\, k_{r1}$, as shown in Figure 7. If more than two links are used, $k_{r\,even} = -2\, k_{r1}$ and $k_{r\,odd} = 2\, k_{r1}$.

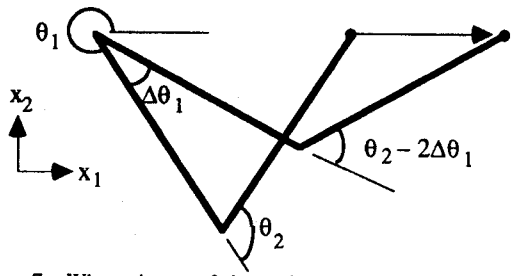


Figure 7. When $k_{r\,2} = -2\, k_{r1}$, the arm extends while keeping x_2 constant. With more links, we would find a characteristic "accordion" shape.

The use of the reflex term gives a significant improvement in the number of iterations needed to move the arm endpoint: I_A = 696 and I_B = 4,343. Performance is unimproved near the origin, as shown by a large I_B; however, the iterations needed for movement in the rest of the workspace are reduced by 85.5%. The major improvements are along extending/retracting lines.

The second obstacle in this study is poor performance near the origin. Since r_1 is small here, $\Delta\theta_1$ is also small. Therefore, many iterations are required to move the endpoint. Also, if the arm is folded up so that the endpoint is near the origin, a small change in endpoint causes a large change in the joint angles.

The easiest way to fix this is to divide $\Delta\Theta$ by a function r_i' which is very small near the origin. The simplest of these functions and the most obvious choice is $|r_i|$. This normalizes the functions. The cross-product term becomes

$$\Delta\theta_{ci} = k_{ci}\frac{|r_i|}{r_i'}|e|\sin(\phi_i) = k_{ci}|e|\sin(\phi_i) \qquad (8)$$

and the reflex term becomes

$$\Delta\theta_{ri} = k_{ri}\frac{|r_i|}{r_i'}|e|\cos(\phi_1) = k_{ri}|e|\cos(\phi_1) \qquad (9)$$

where $r_i' = |r_i|$. These are summed together:

$$\Delta\theta_i = \Delta\theta_{ci} + \Delta\theta_{ri} \qquad (10)$$

where $\Delta\theta_{ri}$ is a function of $\Delta\theta_{r1}$, as discussed earlier in this section. Table 1 is a table of I_A (iterations throughout the workspace in one quadrant) and I_B (iterations near the origin) for all cases mentioned so far.

Case	k_{r1}	k_{c1}	k_{c2}	scaling	I_A	I_B
A	0	4	4	none	4800	3372
B	0	2	2	normalized	3232	90
C	5.5	3	3	none	696	4343
D	1	2	2	normalized	575	88

Table 1. Iterations to convergence as a function of scaling and gains. Normalization and the use of reflex gain $k_{ri} \neq 0$ decrease iteration count.

In each case, the highest gains that did not give more than one overshoot anywhere in the workspace were used. For example, the normalization in case B required lower gains than case A, yet the iteration counts were lower in case B because of the large improvements from normalization. Use of the reflex term leads to fewer iterations throughout the workspace along extending/retracting lines. The normalized scaling greatly improves performance near the origin, as shown by the small values of I_B.

2.4 - *Further Improvements*

The iteration counts may be lowered even further. This can be accomplished using different gains for each angle and using improved scaling. Neural networks may be introduced here as a method of determining these gains.

The best performance is to be obtained when the cross-product terms for angles $\theta_2, \theta_3, \ldots, \theta_n$ are as small as possible, since θ_1 efficiently controls the direction of the entire arm, and the reflex terms control its length. However, at joint limits, there is no reflex term; it is "turned off" by setting all k_{ri} to zero when joint limits are reached because of poor behavior of the algorithm. This agrees with biology, since there is no reflex action when the joints are in these positions. Here, cross-product terms from all angles are needed. This problem can be solved by keeping the cross-product gain for $\theta_2, \theta_3, \ldots, \theta_n$ smaller than the gain of θ_1, yet not zero. Using the normalized version of this algorithm containing both cross-product and reflex terms, there is a definite improvement. Comparing the case when all cross-product gains were identical with the case when $k_{c1} > k_{c2}$, it is shown that the latter case gives better performance (Table 2).

Case	k_{r1}	k_{c1}	k_{c2}	scaling	I_A	I_B
D	1	2	2	normalized	575	88
E	2	2.5	0	normalized	337	71
F	1.8	2.4	0.6	normalized	372	74

Table 2. Iterations to convergence as a function of scaling and gains. The use of different cross-product gains k_{c1} and k_{c2} decrease iteration counts I_A and I_B.

In all cases, the gains k_c and k_r are as large as possible without producing significant oscillations. Case E provides the lowest iteration counts, but does not converge at joint limits. In case F, k_{c2} is much smaller than k_{c1}, so it approximates the ideal case E while converging everywhere. Since $\Delta\theta_{c2}$ is small, case F is more stable than case D, which allows the gains k_c and k_r to be increased. Increasing these gains results in fewer iterations needed for convergence, and the nonzero k_{c2} gives good performance outside arm's reach. This is supported by our results.

Another possible improvement lies in the area of scaling. Normalization may not be the optimal method, although it gives a significant improvement to the algorithm. Other r_i' terms may be superior; however, it is likely that these terms would necessitate longer computation time. Ideally, we want to know the optimal gains for the normalized algorithm. Fewer calculations are required if all we need is a gain instead of a specialized function r_i'. It is also true that each state responds better to a different gain. We can implement neural networks to calculate optimal gains for all states.

2.5 - *Improvements Using Neural Networks*

The algorithm can be improved by using a neural network to find the optimal gains. At this time, neural networks are not used on our system. This is solely an example of how we could implement a simple gradient descent method to optimize our gains.

Using the k_c and k_r from the previous sections as a starting point, we can update the gains as the system is trained throughout the workspace. The system is as follows:

$$\text{state } \Theta(t) = \begin{bmatrix} \theta_1(t) \\ \theta_2(t) \end{bmatrix}$$

$$= \begin{bmatrix} \theta_1(t\text{-}1) + k_{c1}|e|\sin(\phi_1(t\text{-}1)) + k_{r1}|e|\cos(\phi_1(t\text{-}1)) \\ \theta_2(t\text{-}1) + k_{c2}|e|\sin(\phi_2(t\text{-}1)) - 2\,k_{r1}|e|\cos(\phi_1(t\text{-}1)) \end{bmatrix} \qquad (11)$$

$$\text{output } X(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

$$= \begin{bmatrix} l_1\cos(\theta_1(t)) + l_2\cos(\theta_1(t) + \theta_2(t)) \\ l_1\sin(\theta_1(t)) + l_2\sin(\theta_1(t) + \theta_2(t)) \end{bmatrix} \qquad (12)$$

$$\text{weight } w_{ij} = \begin{bmatrix} k_{c1} \\ k_{c2} \\ k_{r1} \end{bmatrix}_{ij} \qquad (13)$$

Subscripts i, j refer to the state of the system. Since we want a relatively small, manageable state matrix, the state will be divided into $p_1 \times p_2$ blocks. The shoulder, with constraints $-180^\circ \le \theta_1 \le 90^\circ$ becomes separated into p_1 categories while the elbow, $0^\circ \le \theta_2 \le 180^\circ$ becomes p_2 categories. w_{ij} is updated only when the state falls into block (i, j). Initially,

$$w_{ij} = \begin{bmatrix} 2.4 \\ .6 \\ 1.8 \end{bmatrix} \qquad (14)$$

for all (i, j). These gains are chosen because they have given the best performance so far.

Minimizing the least squares error energy function

$$E(\Theta,W) = {}^1/_2 [D-X(\Theta,W,t)]^2 \qquad (15)$$

where the desired endpoint position $\mathbf{D} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$ and η is a small positive learning rate, the weight update ΔW is calculated as

$$\Delta W = -\eta \frac{\partial E(\Theta,W)}{\partial W}$$

$$= \eta \begin{bmatrix} \frac{\partial x_1(t)}{\partial k_{c1}} & \frac{\partial x_2(t)}{\partial k_{c1}} \\ \frac{\partial x_1(t)}{\partial k_{c2}} & \frac{\partial x_2(t)}{\partial k_{c2}} \\ \frac{\partial x_1(t)}{\partial k_{r1}} & \frac{\partial x_2(t)}{\partial k_{r1}} \end{bmatrix} \begin{bmatrix} d_1 - x_1(t) \\ d_2 - x_2(t) \end{bmatrix} \qquad (16)$$

The system has been reduced from the two-dimensional plane $\{(\theta_1,\theta_2): -180^\circ \le \theta_1 \le 90^\circ,\ 0^\circ \le \theta_2 \le 180^\circ\}$ to a matrix of dimension $(p_1 \times p_2)$. This simplification has many benefits. The smaller size means fewer calculations have to be performed when updating weights. Also, only $p_1 \times p_2$ positions need to be trained. While updating the weights of one state, the weights of many nearby states are also being updated. Therefore, it is not necessary for the manipulator to visit all states in order to be fully trained.

This reduction of state space has a loss of accuracy associated with it, since a block of state is treated as only one state. A few more iterations might be needed, but we sacrifice this for the speed and simplicity we gain by using a very small state matrix.

We desire a fast, simple method to achieve a desired endpoint. The use of neural networks increases the accuracy by optimizing the gains. This further increases the similarities between our algorithm and biology, since this is now a learning algorithm. This gradient-descent neural network is very basic but learns quickly, due to a reduction in state space.

3 - Application to N-Link Systems

This approach works as well on n-link systems as it does on two-link systems. k_c and k_r must be lowered as n becomes greater, since a small change in each angle produces a progressively larger change in the final endpoint as the number of links increases.

The n-link system has the same problems with retraction / extension as the two-link system had. It is similarly solved by using reflex terms. The reflex gain k_{ri} are as follows: $k_{r\,even} = -2\,k_{r1}$, and $k_{r\,odd} = 2\,k_{r1}$.

Some reflex terms must be turned off when a joint limit is reached, as it was in the two-link case. The method works best when $k_{r2}, k_{r3}, \ldots, k_{rn}$ are all smaller than k_{r1}. Normalization also improves the algorithm greatly, and the response when the endpoint is near the origin is much quicker than when not normalized. The cross-product angle changes may also be weighted in the n-link case. Those joints which do not perform well under high torques can be given smaller weights than others.

Performance for the three-link case is rated in the same way as for two links. Corresponding results are given in Table 3.

Case	k_{r1}	k_{c1}	k_{c2}	k_{c3}	scaling	I_A	I_B
A	0	3.5	3.5	3.5	none	6768	82
B	0	1.5	1.5	1.5	normalized	2399	20
C	7	3	3	3	none	554	103
D	1.5	1.5	1.5	1.5	normalized	497	26
E	1.5	2.5	0	0	normalized	395	∞
F	2	2.5	0.5	0.5	normalized	449	49

Table 3. Iterations to convergence as a function of scaling and gains. Normalization and the use of a reflex gain $k_{ri} \neq 0$ decrease iterations I_A and I_B, as do the use of different cross-product gains k_{c1}, k_{c2} and k_{c3} also decrease I_A and I_B.

In all cases, the highest gains which did not cause significant oscillations were used. As in the two-link case, case E provides the lowest I_A iteration counts. However, it does not function well at joint limits, as shown by I_B. Among those that do converge, case F is optimal.

The same neural network is used on an n-link system. The weight matrix now has n legs and $n+1$ gains for each state (n cross-product gains and 1 reflex gain). As more links are added, the weight matrix increases exponentially. For this reason, fewer state space categories may be wanted. As in the two-link case, only the weights that correspond to the current state are updated.

The method outlined for the two-link case is easily generalizable for the n-link case, since all the problems in both cases are solved using identical means, and improvements to both are received equally well.

4 - Conclusion

Following biological principles, we have developed a means of bringing an n-link arm to an endpoint. As in biology, the joint movements can be computed in parallel. This method of successive approximation is biologically inspired. The joint velocities can be effectively modeled as $\Delta\theta_i$ over a given time, where $\Delta\theta_i$ is the sum of two terms:

$$\Delta\theta_i = \Delta\theta_{ci} + \Delta\theta_{ri} \qquad (10)$$

The actual speed of implementation depends upon the timestep that is used to produce velocities. The normalized cross-product term is

$$\Delta\theta_{ci} = k_{ci}|e|\sin(\phi_i) \qquad (8)$$

and the normalized reflex term is

$$\Delta\theta_{ri} = k_{ri}|e|\cos(\phi_i) \qquad (9)$$

The addition of the reflex term and the use of normalization have been shown to significantly improve the algorithm. The number of iterations needed to obtain a certain

endpoint has decreased and the speed of response has increased. When k_{ci} $(i = 2, 3,..., n)$ are less than k_{c1}, there is also improvement. Further improvements include the use of neural networks to determine the optimal gains, which should result in even better performance.

The method used is very simple to implement and is easily generalizable to an n-link manipulator. This offers an advantage over other methods. It also has the advantage of being inherently compatible with parallel processing: all of the joint angles are computed simultaneously, since the calculations are independent of one another. Unlike some other methods, it is not necessary to compute the entire path before any movement occurs.

This biologically-inspired algorithm works very well for endpoint position control. It may also be applied to force and hybrid control. We are currently working on the dynamic version. This method has been implemented in the control of our redundant manipulator, SLIM, with very promising results.

Acknowledgments

We would like to thank Stephen Lane, David Handelman and Jack Gelfand for their helpful discussions and insights.

References

[1] Nahon, M and J. Angeles. Reducing the Effects of Shocks Using Redundant Actuation. In: *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, April 1991, pp. 238 - 243.

[2] Maciejewski, A. A. and C. A. Klein. Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments. In: *The International Journal of Robotics Research*, Vol. 4, No. 3, Fall 1985, pp., 109 - 117.

[3] Kircanski, M. and M. Vukobratovic. Contribution to Control of Redundant Robotic Manipulators in an Environment with Obstacles. In: *The International Journal of Robotics Research*, Vol. 5, No. 4, Winter 1986, pp. 112 - 119.

[4] Podhorodeski, R. P., A. A. Goldenberg and R. G. Fenton. Resolving Redundant Manipulator Joint Rates and Identifying Special Arm Configurations Using Jacobian Null Spaces. In: *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 5, October 1991, pp. 607 - 618.

[5] Nakamura, Y., H. Hanafusa and T. Yoshikawa. Task-Priority Based Redundancy Control of Robot Manipulators. In: *The International Journal of Robotics Research*, Vol. 6, No. 2, Summer 1987, pp. 3 - 15.

[6] Hollerbach, J. M. and K. C. Suh. Redundancy Resolution of Manipulators through Torque Optimization. In: *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 4, August 1987, pp. 308 - 316.

[7] Maciejewski, A. A. and C. A. Klein. Numerical Filtering for the Operation of Robotic Manipulators through Kinematically Singular Configurations. In: *Journal of Robotic Systems*, Vol. 5, No. 6, 1988, pp. 527 - 552.

[8] Colbaugh, R., H. Seraji and K. L. Glass. Obstacle Avoidance for Redundant Robots Using Configuration Control. In: *Journal of Robotic Systems*, Vol. 6, No. 6, 1989, pp. 721 - 744.

[9] Berkinblit, M. V., I. M. Gel'fand and A. G. Fel'dman. Model of the control of the movements of a multijoint limb. In: *Biophysics*, Vol. 31, No. 1, 1986, pp. 142 - 153.

[10] Hinton, G. Parallel Computations for Controlling an Arm. In: *Journal of Motor Behavior*, Vol. 16, No. 2, 1984, pp. 171 - 194.

[11] Zhang, Hong, Paul, Richard P. A Parallel Inverse Kinematics Solution for Robot Manipulators Based on Multiprocessing and Linear Extrapolation. In: *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 5, 1991, pp. 660 - 669.

[12] Kreyszig, Erwin. *Advanced Engineering Mathematics, Sixth Edition*. John Wiley & Sons, 1988.