'89 KACC 1989. 10. 27~28

A Computed Torque Method Incorporating an Iterative Learning Scheme

Kwanghee Nam

Department of Electrical Engineering, POSTECH,
Hyosa San-31, Pohang, Kyungbuk 790-600, R. of Korea

Abstract: An iterative learning control scheme is incorporated to the computed torque method as a means to enhance the accuracy and the flexibility. A learning rule is constructed by utilizing a gradient descent algorithm and data compressing techniques are illustrated. Computer simulation results show a good performance of the scheme under a relatively high speed and a heavy payload condition.

1 Introduction

Since the dynamics of robot manipulators is highly nonlinear, it is not easy with classical fixed gain controllers to accomodate the parameter changes resulting from nonlinear charateristics. As a nonlinearity compensating controller, the computed torque method[3] was proposed. However, the computed torque method not only requires apriori knowledge of robot dynamics but also computation time at each sample point. Further, if there are some unexpected errors such as parameter errors and the payload change, then the controller using the computed torque method becomes rather sensitive. Various techniques of model following adaptive control[4] have been utilized to incompass such an inflexibility without a precise description of the dynamic model. A general drawback of adaptive approach is a large computational load for the real-time parameter identification.

Recently, iterative learning techniques[2,6] based on the repeatability of robot motion have been paid a special attention as a new control scheme. Apart from the simplicity and straightforwardness, the iterative learning methods are flexible because they do not require the exact dynamic model. Miller et. al.[6] have proposed a feasible control scheme utilizing *Cerebellar Model Articulation Controller (CMAC)* which was developed by Albus[1]. Following the work of Miller III et.al.[6], T. Kuc and K. Nam[5] also utilized a version of CMAC memory for a learning control of the robot motion without acceleration measurements.

In this work, a leaning control scheme is incorporated to the computed torque method in order to overcome the problems associated with computed torque method such as the sensitivity and the inflexibility. In section 2, a computed torque method is illustrated. In section 3, a learning rule is derived by applying a gradient descent algorithm to an index function. A global learning control scheme is shown in section 4 and data storage techniques are shown in section 5. Finally, computer simulation results of an example are shown in section 6.

2 Computed Torque Method

Throughtout the paper, we consider the following dynamics of a robot having the n-degree of freedom:

$$D(q)\ddot{q} + C(q,\dot{q}) + g(q) = \tau, \tag{1}$$

where $q \in R^n$ is a vector of joint angles and τ is a vector of torques applied to each joint. Here, the matrix $D(q)$ is called inertia matrix, $C(q,\dot{q})$ represents Coriolis force plus Coulomb fricition and g is a gravity vector. Together with (1), we consider the forward kinematic equation:

$$z = G(q) \tag{2}$$

where $z \in R^n$ denote the postion in a fixed task-related Cartesian coordinates and the map $G : R^n \rightarrow R^n$ relates the joint coordiantes to the Cartesian coordinates. For general six degree of freedom manipulator, z contains 3 position components and 3 orientation components. Here, we consider the system (1) in an open subset O of R^n in which $D(q)$ is nonsigular and the map $G : O \rightarrow G(O)$ is one to one. In other words, we consider the robot system in a subset where no sigular configuration occurs. We assume that the maps D, C, g are smooth functions of their variables.

Since there are incidental attributes in the fabrication of robot links, inertial parameters of links such as mass, center of mass, moments of inertia tend not to be known exactly. Furthermore, the fucntions $D(q)$,$C(q,\dot{q})$, $g(q)$ are dependent on the change of payloads. For these reasons we assume that the exact quantities of $D(q)$,$C(q,\dot{q})$, $g(q)$ are not known. We denote the estimate of $D(q)$,$C(q,\dot{q})$, $g(q)$ by $\hat{D}(q)$,$\hat{C}(q,\dot{q})$, $\hat{g}(q)$ and let

$$\begin{aligned}\Delta D(q) &= \hat{D}(q) - D(q)\\ \Delta C(q,\dot{q}) &= \hat{C}(q,\dot{q}) - C(q,\dot{q})\\ \Delta g(q) &= \hat{g}(q) - g(q).\end{aligned}$$

Following the idea of computed torque method, we choose the following input torque based on the estimates $\hat{D}$, $\hat{C}$, $\hat{g}$:

$$\tau = \hat{D}(q)\{\ddot{q}_d - K_v(\dot{q}-\dot{q}_d) - K_p(q-q_d)\} + \hat{C}(q,\dot{q}) + \hat{g}(q), \quad (3)$$

where q_d denotes a desired trajectory. Then, letting $e = q - q_d$, we obtain an error equation:

$$\ddot{e} + K_v\dot{e} + K_p e = E(t) \quad (4)$$

where K_v, K_p are constant diagonal matrices and

$$E(t) = \hat{D}^{-1}(q)(\Delta D(q)\ddot{q} + \Delta C(q,\dot{q}) + \Delta g(q)). \quad (5)$$

If we konw the perfect information about the robot parameters, i.e., if there is no modeling errors, then the driving term $E(t)$ in (4) vanishes. Obviousely, K_v, K_p must be chosen such that the error system is stable. For the stability of the error system (4), K_v and K_p must be sufficiently large in order to dominate the effects of the nonlinear perturbing term $E(t)$ which may cause instability. It is this reason why most of robot controllers utilize high gain feedback. Although it is possble to make robot system stable, there still remains trajectory error along the trajectory unless the nonlinear error term $E(t)$ disappear completely. For the elimination of such a mismatch error $E(t)$, we add a compensating torque $-\hat{D}(q)H^i(t)$ with an objective of canceling out the nonlinear term $E(t)$ such that

$$\begin{aligned}\tau(t) &= \hat{D}(q)\{\ddot{q}_d - K_v(\dot{q}-\dot{q}_d) - K_p(q-q_d)\}\\ &\quad +\hat{C}(q,\dot{q}) + \hat{g}(q) - \hat{D}(q)H^i(t). \quad (6)\end{aligned}$$

Then the error equation becomes

$$\ddot{e} + K_v\dot{e} + K_p e = E(t) - H^i(t). \quad (7)$$

Since the nonlinear mismatch error $E(t)$ is unkown, $H^i(t)$ should be trained so that for $i \to \infty$

$$H^i(t) \to E(t). \quad (8)$$

By superscript i of H^i, we denote the iteration number. The term $H^i(t)$ will be updated by a learing rule which will be shown in the next section.

3 A Learning Rule

Since $(q,\dot{q})$ are measured at each sample period and the input torque τ is also applied at the same periodicity, we put an integer variable k instead of t such that

$$\begin{aligned}\tau(k) &= \hat{D}(q)\{\ddot{q}_d - K_v(\dot{q}-\dot{q}_d) - K_p(q-q_d)\}\\ &\quad +\hat{C}(q,\dot{q}) + \hat{g}(q) - \hat{D}(q)H^i(k). \quad (9)\end{aligned}$$

As was mentined in the previous section, the goal of the learning is to let $H^i(k) \to E(k)$ for each k as the iteration proceeds. For the derivation of the learning rule, we consider the following index:

$$J_k = \frac{1}{2}\sum_{i=1}^{\infty} ||E(k) - H^i(k)||^2 \quad (10)$$

Applying gradient descent rule, we obtain

$$\begin{aligned}H^{i+1}(k) &= H^i(k) - \beta\frac{\partial J_k}{\partial H^i(k)},\\ &= H^i(k) + \beta(E(k) - H^i(k)), \quad (11)\end{aligned}$$

where β is a positive constant and it is often called a ***training factor***. Since $E(k)$ is not available, (11) cannot be used as a learning rule. However, if we neglect the acceleration term $\ddot{e}^i(t)$ assuming that the motion of robot is smooth and K_v and K_p are sufficiently large, it follows from (7) that

$$E(k) - H^i(k) \simeq T_e(k), \quad (12)$$

where

$$T_e(k) = K_v\dot{e}(k) + K_p e(k). \quad (13)$$

Utilizing (13) we obtain the learning rule which is driven by known values:

$$H^{i+1}(k) = H^i(k) + \beta T_e^i(k). \quad (14)$$

For an initial condition we let $H^1(k) = 0$ for $k = 1, 2, \cdots$.

4 A Learning Controller

We call it a learning control scheme a controller in which a learning algorithm is involved. The learning control scheme of this paper is decribed by the torque input (9) with the the update rule (14). This controller differs from the computed torque method in the sense that $\hat{D}(q)H^i(k)$ is added with a purpose of cancelling out the nonlinear mismatch error $E(k)$, and that $H^i(k)$ is updated iterationwise for each k.

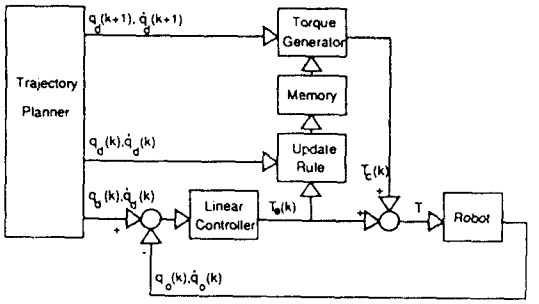

Figure 1. A learning control scheme for a manipulator.

Figure 1 shows a block diagram of learning control scheme utilized in this work. In this control scheme, the linear control part $\ddot{q}_d + K_v(\dot{q} - \dot{q}_d) + K_p(q - q_d)$ of (9) plays a role in attracting the state of the system to the stable region, from which the learning rule makes a precise tuning to the desired output. The torque $\hat{D}(q)H^i(k)$, namely a compensating torque, is generated from a learning controller to compensate the uncanceled nonlinear term which may be caused by the uncertainty of robot parameters. In the learning controller, the compensating torques $H^i(k)$ are updated for the next iteration at each step. Since the position and velocity errors are large at the initial stage of learning, the torque $T_e^i(k)$ from the linear controller is not small initially while the torque $\hat{D}(q)H^i$'s are set to zero. Thus, initially the torque T_e^i is dominant over the compensating torque $\hat{D}(q)H^i$. Conversely, if position and velocity errors are small after a sufficient learning, $T_c^i(k)$ becomes more significant than T_e^i. In an extreme case where there are no trajectory errors, i.e., $e^i(k) = 0$, the torque $T_e^i(k)$ vanishes, while the following holds

$$H^i(k) = E(k). \tag{15}$$

5 Data Storage Techniques

5.1 A simple method of reducing memory size

For the implementation of the previous learning rule, $\{H^i(k)\}_{k=1}^n$, n being a final step, need to be stored for the update. Since the number of steps grow with the length of a trajectory, the memory size for $\{H^i(k)\}_{k=1}^n$ also increases with the length of a trajectory. Thus, the memory size may become a bottle neck in implementing a learning controller for a fairly long trajectory. In this section, a method of reducing the data storage size is suggesed.

Let $\Delta H^i(k) = H^i(k+1) - H^i(k)$. Storing $\{H^i(k)\}_{k=1}^n$ is equivalent to storing the differences $\{\Delta H^i(k)\}_{k=1}^{n-1}$ with the initial value $H^i(1)$, because one can reconstruct $H^i(k)$ in a recursive manner such that

$$H^i(k+1) = H^i(k) + \Delta H^i(k). \tag{16}$$

But, we claim that storing $\{\Delta H^i(k)\}_{k=1}^{n-1}$ with $H^i(1)$ has a significant advantage over storing $\{H^i(k)\}_{k=1}^n$ in the perspective of minimizing the memory size, because $|\Delta H^i(k)|$'s are much smaller than $|H^i(k)|$'s except the cases where $H^i(k)$'s are near zero. That is, one can save the memory space by storing the difference $\{\Delta H^i(k)\}_{k=1}^{n-1}$ which are smaller than $\{H^i(k)\}_{k=1}^n$, since small numbers can be expressed by a small number of data bits. For example, chances are that only 8 bits are required for the storge of $\Delta H^i(k)$'s although 12 bits are required for the storage of $H^i(k)$.

Now, the question is how close the neighboring values $H^i(k+1)$ and $H^i(k)$ are. The target value of $H^i(k)$ is $E(k)$ which is a continuous function of $q(k)$, $\dot{q}(k)$, and $\ddot{q}(k)$. Thus, one may regard $H^i(k)$'s as a continuous function of $q(k)$, $\dot{q}(k)$, and $\ddot{q}(k)$, since $H^i(k) \simeq E(k)$ in a neighborhood of the desired trajectory q_d. Note further that the differences of q, $\dot{q}$, $\ddot{q}$ over the sample period, i.e., $q(k+1) - q(k)$, $\dot{q}(k+1) - \dot{q}(k)$, $\ddot{q}(k+1) - \ddot{q}(k)$ are small if either the robot motion is slow or sampling frequency is high. However, in most cases of robot control the sampling frequency is sufficiently high. Otherwise, nonlinear behaviour becomes dominant so that it causes a large trajectory error. Therefore, $\Delta H^i(k)$'s will become small.

5.2 A CMAC based mapping rule

In order to prevent the memory size from growing large indefinitely, a version of CMAC memory is utilized in [5] in which memory is used as associative mapping elements and its contents are adjusted by learning rule (14). An advantage of utilizing this kind of technique is to reduce the memory size in storing input and output data of a continuous function. In the following, the mapping rule suggested in [5] is illustrated.

Since the values of a continuous function for neighboring points are not so much different, one may be able to save the size of memory in storing the output values of a function by sharing a portion of the output data corresponding to two adjacent input data. Note that the CMAC is basically a table look-up control technique in which a complex nonlinear function is represented by a distributed data form to be added. For more specific illustration of CMAC, refer to [6,1].

In this section, we describe a mapping rule (or partitioning rule) for the data storage by utilizing the concept of content addressing. Note from (5) that $E(k)$ is a function of $q(k)$, $\dot{q}(k)$, $\ddot{q}(k)$. Since $H^i(k)$ can be also regarded as a function of $q_d(k)$, $\dot{q}_d(k)$, $\ddot{q}_d(k)$ in a neighborhood of the desired trajectory q_d, we may store the value of $H^i(k)$ to the place addressed by $q_d(k)$, $\dot{q}_d(k)$, $\ddot{q}_d(k)$. Since the desired trajectories are obtained

apriori, the values of $q_d(k)$, $\dot{q}_d(k)$, $\ddot{q}_d(k)$ are always available.

Suppose that $q_d(k)$, $\dot{q}_d(k)$ and $\ddot{q}_d(k)$ are represented by l digits floating point numbers such as for $1 \le k \le n$

$$\begin{aligned} q_d(k) &= \delta_a^k \bar{a}_1^k \bar{a}_2^k \cdots \bar{a}_l^k, \\ \dot{q}_d(k) &= \delta_b^k \bar{b}_1^k \bar{b}_2^k \cdots \bar{b}_l^k, \\ \ddot{q}_d(k) &= \delta_c^k \bar{c}_1^k \bar{c}_2^k \cdots \bar{c}_l^k, \end{aligned}$$

where δ_a^k, δ_b^k and δ_c^k denote the signs of $q_d(k)$, $\dot{q}_d(k)$ and $\ddot{q}_d(k)$ respectively, and fixed points are not specified. The digits, $\bar{a}_j^k$, $\bar{b}_j^k$ and $\bar{c}_j^k$ for $1 \le j \le l$ may have an arbitray representation, but we assume that they are represented by decimal numbers. To avoid conflict in the following addressing technique, we let for $1 \le j \le l$

$$a_j^k = \delta_a^k \bar{a}_j^k, \qquad b_j^k = \delta_b^k \bar{b}_j^k, \qquad c_j^k = \delta_c^k \bar{c}_j^k. \tag{17}$$

In order words, we define a_j^k, b_j^k, c_j^k by putting the same signs on $\bar{a}_j^k$, $\bar{b}_j^k$ and $\bar{c}_j^k$ as those of the original data, $q_d(k)$, $\dot{q}_d(k)$, $\ddot{q}_d(k)$. The reason is to address two data whose adresses are the same except signs to different memory locations.

Since the points $q_d(k+1)$, $\dot{q}_d(k+1)$, $\ddot{q}_d(k+1)$ are next to the point $q_d(k)$, $\dot{q}_d(k)$, $\ddot{q}_d(k)$, some portion of significant bits of them may be the same, i.e., for most cases there exist nonzero numbers $m_a^k, m_b^k, m_c^k > 0$ such that

$$\begin{cases} a_j^k = a_j^{k+1}, & 1 \le j \le m_a^k \\ b_j^k = b_j^{k+1}, & 1 \le j \le m_b^k \\ c_j^k = c_j^{k+1}, & 1 \le j \le m_c^k. \end{cases}$$

If the signs of adjacent two data with nonzero digits are different, then either $m_a^k = 0$, $m_b^k = 0$, or $m_c^k = 0$.

Example 1: Suppose that we have $q_d(k) = 3.468$, $\dot{q}_d(k) = -0.377$, and $\ddot{q}_d(k) = 0.0$. Then $(a_1^k, b_1^k, c_1^k) = (3, 0, 0)$, $(a_2^k, b_2^k, c_2^k) = (4, -3, 0)$, $(a_3^k, b_3^k, c_3^k) = (6, -7, 0)$ and $(a_4^k, b_4^k, c_4^k) = (8, -7, 0)$. Further, if $q_d(k+1) = 3.467$, $\dot{q}_d(k+1) = -0.381$ and $\ddot{q}_d(k+1) = -0.001$, then $m_a^k = 3$, $m_b^k = 2$ and $m_c^k = 3$.

Hereafter, we use the four dimensional vector (a_j^k, b_j^k, c_j^k, j) as an addressing unit and call the memory location addressed by (a_j^k, b_j^k, c_j^k, j) *memory cell*. We will store the value of $H^{i+1}(k)$ in the l memory locations, (a_j^k, b_j^k, c_j^k, j), $1 \le j \le l$ after partitioning $H^{i+1}(k)$. We let $m_d^k = min\{m_a^k, m_b^k, m_c^k\}$. Then, since $(a_j^k, b_j^k, c_j^k) = (a_j^{k+1}, b_j^{k+1}, c_j^{k+1})$ for $1 \le j \le m_d^k$, for $m_d^k > 0$, m_d^k memory cells must be shared in storing $H^{i+1}(k)$ and $H^{i+1}(k+1)$. Now, it remains to show how to partition and store the values of $H^{i+1}(k)$ and $H^{i+1}(k+1)$ so that the m_d^k mamory cells are shared for $k = 1, 2, \ldots, n$. For this purpose, we will define the following k-depnedent recursive (patitioning) map $F^{(k,i)} : R^{3l} \to R^l$ and will save the value $F_j^{(k,i+1)}(a_j^k, b_j^k, c_j^k)$ in the memory location (a_j^k, b_j^k, c_j^k, j), where $F_j^{(k,i+1)}$ implies the j-th component of $F^{(k,i)}$.

For $k = 1$, we let

$$F_j^{(1,i+1)}(a_j^1, b_j^1, c_j^1) = F_j^{(1,i)}(a_j^1, b_j^1, c_j^1) + \frac{\beta}{l} T_e(1). \tag{18}$$

For $(k, i) \ne (1, i)$,

$$F_j^{(k+1,i+1)}(a_j^{k+1}, b_j^{k+1}, c_j^{k+1})$$

$$= \begin{cases} F_j^{(k,i+1)}(a_j^k, b_j^k, c_j^k), & 1 \le j \le m_d^k \\ \\ F_j^{(k+1,i)}(a_j^{k+1}, b_j^{k+1}, c_j^{k+1}) \\ +\frac{1}{l-m_d^k}(\beta T_e(k+1) \\ -\sum_{r=1}^{m_d^k}\{F_j^{(k,i+1)}(a_j^k, b_j^k, c_j^k) \\ -F_j^{(k+1,i)}(a_j^{k+1}, b_j^{k+1}, c_j^{k+1})\}\Big) & m_d^k + 1 \le j \le l \\ . \end{cases} \tag{19}$$

For an initialization, we let $F_j^{(k,1)} = 0$ for $1 \le k \le n$.

It follows directly from (14) that

$$H^{i+1}(k) = \beta \sum_{r=1}^{i} T_e(k). \tag{20}$$

Bearing (20) in mind and utilizing (18,19), one can easily check the following identity:

$$H^{i+1}(k) = \sum_{j=1}^{l} F_j^{(k,i+1)}. \tag{21}$$

This implies that $H^{i+1}(k)$ can be recovered by summing the contents $F_j^{(k,i+1)}$'s in memory cells (a_j^k, b_j^k, c_j^k, j) for $1 \le j \le l$.

Example 2: We let $q_d, \dot{q}_d, \ddot{q}_d$ be the same as in example 1 for $k > 1$. Assume that $H^{i+1}(k) = -20.0$ and its value is stored in a distributed form such as

$$\begin{bmatrix} F_1^{(k,i+1)}(3,0,0) \\ F_2^{(k,i+1)}(4,-3,0) \\ F_3^{(k,i+1)}(6,-7,0) \\ F_4^{(k,i+1)}(8,-7,0) \end{bmatrix} = \begin{bmatrix} -6 \\ -5 \\ -3 \\ -6 \end{bmatrix}.$$

Similarly, we assume that $H^i(k+1) = -12.0$ is stored such that

$$\begin{bmatrix} F_1^{(k+1,i)}(3,0,0) \\ F_2^{(k+1,i)}(4,-3,0) \\ F_3^{(k+1,i)}(6,-8,0) \\ F_4^{(k+1,i)}(7,-1,-1) \end{bmatrix} = \begin{bmatrix} -5 \\ -4 \\ -2 \\ -1 \end{bmatrix}.$$

Since $m_d^k = 2$, two components of $F^{(k,i+1)}$ and $F^{(k+1,i+1)}$ must be the same, i.e., $F_1^{(k+1,i+1)} = F_1^{(k,i+1)} = -6.0$, $F_2^{(k+1,i+1)} = F_2^{(k,i+1)} = -5$. Assuming that $\beta = 1.0$ and $T_e(k+1) = -4.0$, we obtain from (14) that $F_3^{(k+1,i+1)}(6,-8,0) = -3.0$ and $F_4^{(k+1,i+1)}(7,-1,-1) = -2$. Further, one can easily check the following identity:

$$\begin{aligned} H^{i+1}(k+1) &= \sum_{j=1}^{4} F_j^{(k+1,i+1)} = -16.0 \\ &= H^i(k+1) + \beta T_e(k+1) \end{aligned}$$

One can notice at this point why we put the same sign before each digit as that of the original data like (17).

In most cases, the contribution of the torque caused by acceleration to robot dynamics are relatively small. Further, if the motion of robot along the desired trajectory is sufficiently smooth, then the effect of acceleration term $\ddot{q}_d$ to the dynamics becomes negligible. In this case, we can further optimize the memory size by eliminating the acceleration term in addressing.

6 A Simulation Example and Results

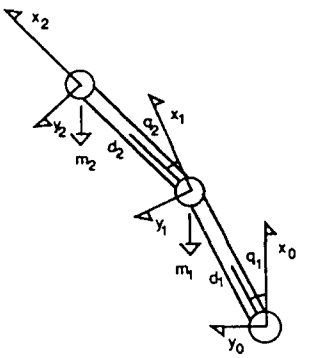

Figure 2. Schematic diagram of a two-axis robot manipulator.

For a simulation example, we consider the above two-axis robot manipulator. m_i, d_i and q_i represent the mass, length and joint angle of the link i for $i = 1, 2$ and m_p denotes a payload. The dynamic model of the manipulator is given by

$$T_1 = D_{11}\ddot{q}_1 + D_{12}\ddot{q}_2 + h_{122}\dot{q}_2^2 + h_{112}\dot{q}_1\dot{q}_2 + G_1 \quad (22)$$

$$T_2 = D_{21}\ddot{q}_1 + D_{22}\ddot{q}_2 + h_{211}\dot{q}_1^2 + G_2, \quad (23)$$

where

$$D_{11} = [(M_1 + M_2)d_1^2 + M_2 d_2^2 + 2M_2 d_1 d_2 \cos q_2],$$

$$D_{12} = D_{21} = M_2 d_2^2 + M_2 d_1 d_2 \cos q_2,$$

$$D_{22} = M_2 d_2^2,$$

$$h_{112} = -2M_2 d_1 d_2 \sin q_2,$$

$$h_{122} = -h_{211} = -M_2 d_1 d_2 \sin q_2,$$

$$G_1 = -[(M_1 + M_2)g d_1 \sin q_1 + M_2 g d_2 \sin(q_1 + q_2)],$$

$$G_2 = -M_2 g d_2 \sin(q_1 + q_2),$$

$$M_1 = m_1, \qquad M_2 = m_2 + m_p,$$

We let $m_1 = 10Kg$, $m_2 = 10Kg$, $d_1 = 1.0m$ and $d_2 = 1.0m$. For desired joint trajectories, we choose for $0 \le t \le 4$,

$$\begin{bmatrix} q_{d1}(t) \\ q_{d2}(t) \end{bmatrix} = \begin{bmatrix} (\pi/3)\cos(2\pi/5)t \\ -(\pi/2)\cos(2\pi/5)t - \pi/4 \end{bmatrix},$$

$$\begin{bmatrix} \dot{q}_{d1}(t) \\ \dot{q}_{d2}(t) \end{bmatrix} = \begin{bmatrix} -(2\pi^2/15)\sin(2\pi/5)t \\ (\pi^2/5)\sin(2\pi/5)t \end{bmatrix}.$$

Further, we choose $\beta = 1.0$ for joints 1, 2, and $0.002sec$ for the sampling time interval, and $K = 30$, $L = 15$. The actual trajectories were obtained by solving differential eqautions (22,23) with Runge-Kutta 4th method.

The payload m_p is chosen as $30kg$, but it is treated as a perturbation to the robot system. That is, the quantities $\hat{D}(q)$, $\hat{C}(q,\dot{q})$, $\hat{g}(q)$ are calculated for $m_p = 0$, while it is assumed that the actual robot manipulator has a payload of $30Kq$. This makes an error for a computed torque input and, as a result, produces trajectory errors which are shown in Figures 3,4. That is, Figures 3,4 show the actual trajectories of the system before learning along with the desired trajectory. Figures 5,6 show that the trajectories of the same system which have converged to the desired ones after 30th iteration.

For a data storage technique, we utilized a simple method in section 5.1. Figure 7 shows the plots of $H^i(k)$ and $100\Delta H^i(k)$ versus sample points, from which we can observe that $|\Delta H^i(k)|$ is about one hundredth of $|H^i(k)|$. Hence, storing $\Delta H^i(k)$ rather than $H^i(k)$ will save the memory space as was mentioned in section 5.1. If the sampling interval is increased, then the relative magnitude of $|\Delta H^i(k)|$ with respect to that of $|H^i(k)|$ will also increases.

There is no need to update $H^i(k)$ for each sample point. Conversely, if the update is made every m step for some $m > 1$, then one can save the memory size for the learning. But, the interval between the update points grows longer, then the learning scheme may not work satisfactorily. Hence, there may be a trade-off between the performance of a learning scheme and the memory size for learning.

7 Conclusion

A learning control scheme is added to the computed torque method so that the trajectory errors vanish. But, learning control schemes require large memory size for the update, thus techniques of reducing memory size are important. There need to be further studies on the data compressing or addressing techniques and also on the choice of the data update interval. A good performance of this control scheme was exemplied through a computer simulation.

References

[1] J. S. Albus,, "Brains, Behavior, and the Robotics" N. H. Byte Books, 1981.

[2] S. Arimoto, S. Kawamura, and F. Miyasaki, "Bettering operation of robots by learning," *J. Robotic Syst.*, pp. 123-140, 1984.

[3] A.K.Bejczy, "Robot Arm Dynamics and Control," *Technical Memo 33-669*, Jet Propulsion Laboratory, Pasadena, CA, 1974.

[4] C. S. G. Lee and M.J. Chung, "An adaptive control strategy for mechanical manipulators," *IEEE Trans. on Automat. Contr.*, Vol. AC-29, pp. 837-840, 1984.

[5] T. Kuc and K. Nam, "CMAC based iterative learning control of robot manipulators," 1989 CDC, Tampa, FL.

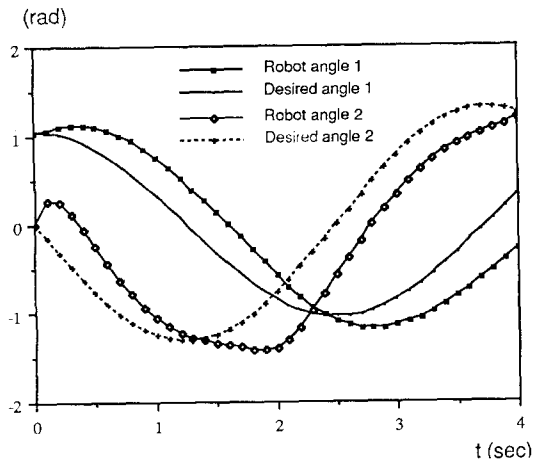

Figure 3. Joint angle trajectories of the system before learning.

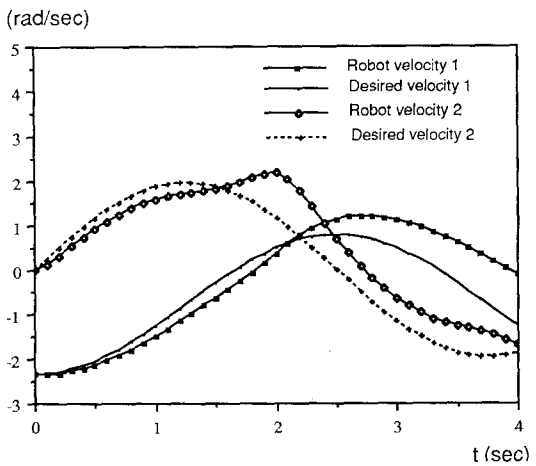

Figure 4. Joint angular velocity trajectories of the system before learning.

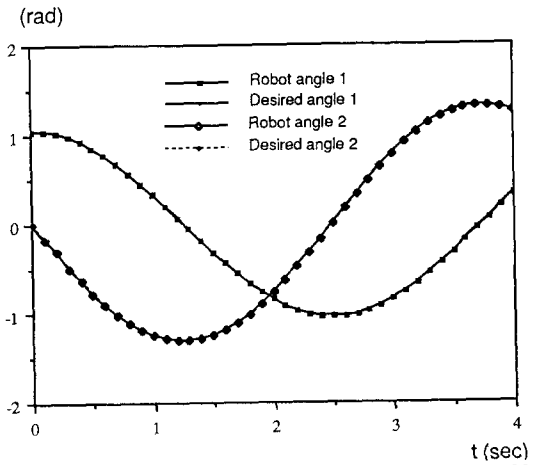

Figure 5. Joint angle trajectories of the system after the 30th iteration.

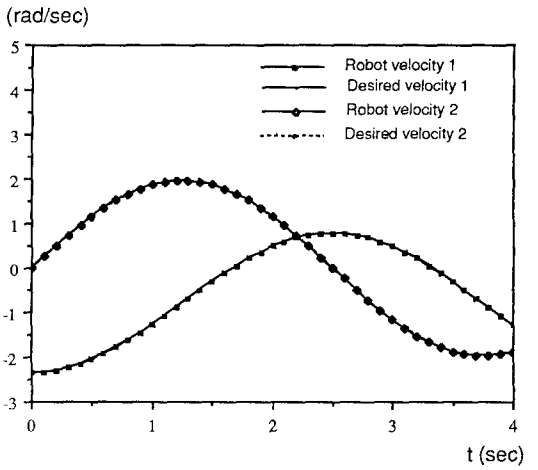

Figure 6. Joint angular velocity trajectories of the system after the 30th iteration.

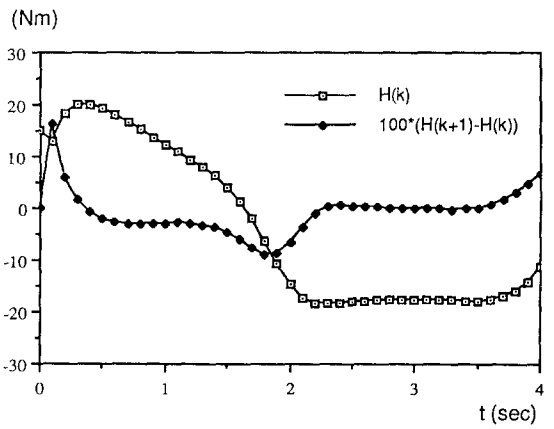

Figure 7. Comparison of $H^i(k)$ and $\Delta H^i(k)$ versus k.