An On-Line Distributed Minimum-Time Trajectory Generator for Intelligent Robot Manipulators

by:

Ali M. S. Zalzala and Alan S. Morris

Department of Control Engineering,

University of Sheffield,

Mappin Street,

Sheffield S1 3JD,

U.K.

Research Report # 358

April 1988

Abstract

An algorithm is presented for the on-line generation of minimum-time trajectories for robot manipulators. The algorithm is designed for intelligent robots with advanced on-board sensory equipment which can provide the position and orientation of the end-effector. Planning is performed in the configuration (joint) space by the use of optimised combined polynomial splines, along with a search technique to identify the best minimum-time trajectory. The method proposed considers all physical and dynamical limitations inherent in the manipulator design, in addition to any geometric path constraints. Meeting the demands of the heavy computations involved lead to a distributed formulation on a multiprocessor system, for which an intelligent control unit has been created to supervise its proper and practical implementation. Simulation results of a proposed casy study are presented for a PUMA 560 robot manipulator.

I: Introduction

The importance of the problem of robot trajectory planning and control is evidenced by the large amount of the related research literature. The difficulties associated with tackling such a problem arise from different inherent properties in the manipulator design. The fact of the nonlinear, coupled nature of the arm dynamics [18] imposes a great computational burden [16,10]. In addition, several limitations are encountered concerning the joint angular velocities and actuator torque (or force) values. The presence of any obstacles in the robot work space would introduce the additional burden of searching for the best path for the end-effector to traverse [15]. Hence, due to all these problematic issues, the tendency has been towards the division of the problem of trajectory control into two sub-tasks, namely: trajectory planning and trajectory tracking. A trajectory planner is executed off-line, where specific manipulator motion has to be described and a suitable space curve for the robot end-effector traversal is to be generated. A time history of position, velocities, accelerations and torques should be supplied to drive the control loops on the robot joints. Several methods had been suggested for planning robot trajectories, for which comprehensive surveys can be found in [21,24].

The general approach in applying off-line planning of a trajectory that would move the robot hand from a *start* point to an *end* point is by the provision of several intermediate-state points (i.e. *via–points*). This is of extreme importance when certain obstacles are known to be present in the robot volume. However, once a successful and suitable trajectory is predicted for the required task, any changes of the proposed via-points or the imposed constraints require the complete re-computation of a new trajectory. Considering the fact that optimal planners would require substantial computation, such an approach is inappropriate in certain industrial applications where different unpredicted tasks are expected to occur in a changing environment [9]. It is the need for an efficient on-line trajectory generator that encouraged the contribution presented by this paper. It is intended to provide an accurate and practical form of robot path tracking in an integrated automated system, with a high level of sensory intelligence [9,19,7].

In this paper, a unique trajectory generator developed for the control of robot manipulators is described. The concept of *point–look–ahead* is introduced in an algorithm designed to accomplish *maximum performance in a minimum of time.* Planning is made on local basis in the joint space by the use of combined polynomial splines. The method proposed considers all physical and dynamical limitations inherent in the manipulator design, in addition to any geometric constraints imposed on the path, thus accommodating easily for any obstacles in the work volume.

Although a heavy computational burden is involved for an on-line application, the fact of the algorithm being highly structured and consisting mainly of repetitive computational blocks that are partialy dependent makes such an implementation possible. Such a structure allows for a high level of parallelism and concurrency in the computations, and a distributed architecture has been developed to accomodate for it.

In the remaining of this paper, the proposed work is presented as follows: In section II, a statement of the problem is defined, while the formulated algorithm is illustrated in section III. The computational complexity of the on-line trajectory planner is discussed in section IV. In section V, a mapping of the algorithm on a multiprocessor system is illustrated, along with the intelligent control unit proposed. Simulation results for a case study involving the PUMA 560 manipulator is included in section VI. Finally, conclusions are made in section VII.

II : Problem Statement

The aim of this work is to construct the minimum-time trajectories in joint-space, while the mechanical manipulator is actually in action performing a certain task. Thus, the minimization of travelling time is the criterion to consider in on-line motion planning.

II.1: *The point–look–ahead (PLA) concept* :

For the robot end-effector to track a minimum-time trajectory using the information provided by its on-board sensory equipment, the proposed trajectory should be constructed on-line with a *progressive segments* method. Hence, while the manipulator hand is traversing one *present* segment, another *next* segment is being computed by the controller, based on the sensors advice. The border point between these two present and next segments of the trajectory is defined as the *look–ahead point.* Each single look-ahead point defines explicitly the position and orientation of the robot hand at that specific moment. Thus, a set of n look-ahead points are sufficient to define the required trajectory. However, since planning is to be performed in the manipulator joint space, each position and orientation of the end-effector must be transformed from the hand Cartesian coordinates to its corresponding joint values, producing N sets of n points, where N is the number of degrees-of-freedom of a given manipulator, namely

$$\boldsymbol{\theta}_{ij} \quad , i=1,2,\ldots,n \quad , j=1,2,\ldots,N \tag{1}$$

This is accomplished by applying an Inverse Kinematics algorithm to each position matrix

$$\mathbf{H}(t_i) = \begin{bmatrix} \mathbf{n}(t_i) & \mathbf{s}(t_i) & \mathbf{a}(t_i) & \mathbf{p}(t_i) \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

where,

$$\mathbf{n}(t_i) \equiv \textit{unit normal vector,}$$

$$\mathbf{s}(t_i) \equiv \textit{unit slide vector,}$$

$$\mathbf{a}(t_i) \equiv \textit{approach vector,} \text{ and}$$

$$\mathbf{p}(t_i) \equiv \textit{position vector.}$$

yielding a set of n-length vectors $\boldsymbol{\Theta}_i$, one for each degree-of-freedom, as expressed in (1) [18].

The most important aspect of the *PLA* concept is that the via-points, necessary for the construction of the trajectory, are not chosen by the user priori to planning, but rather selected as the most suitable point whenever needed. This would release the constraint on having a *structured* environment, where all path constraints, including the obstacles (if any), should be pre-organized and, moreover, fixed during the performance of motion.

Nevertheless, since the provided look-ahead points are in discrete values, a continuous history of the trajectory parameters passing through these points should be constructed, which suffices for the continuity required. This will be discussed in section III following.

II.2: *Planning constraints* :

For the planning procedure to be accurate, it should take into consideration all realistic constraints which may limit the manipulator performance [3]. The constraints to be considered by this algorithm are the joints limits on angular position, velocity, acceleration and jerk. The first of these is imposed by the manipulator geometric design, while the second and third limits are defined by the capabalities of each joint actuator. Hence, the following limitations are set:

$$\theta limit_j^- \leq \theta_j(t_i) \leq \theta limit_j^+ \tag{3}$$

$$\dot{\theta}_j(t_i) \leq |\ \dot{\theta} limit_j\ | \tag{4}$$

$$\ddot{\theta}_j(t_i) \leq |\ \ddot{\theta} limit_j\ | \tag{5}$$

$$\dddot{\theta}_j(t_i) \leq |\ \dddot{\theta} limit_j\ | \tag{6}$$

where,

$$\theta limit_j^- = \textit{lower bound on position},$$

$$\theta limit_j^+ = \textit{upper bound on position},$$

$$\dot{\theta} limit_j = \textit{bound on velocity},$$

$$\ddot{\theta} limit_j = \textit{bound on acceleration},$$

$$\dddot{\theta} limit_j = \textit{bound on jerk}.$$

and, $j \in \{1,2,...,N\}$.

One further major constraint to be considered is the coupled and highly nonlinear dynamic equations of motion [18]. Thus , a limit

$$\tau_j(t_i) \leq |\ \tau limit_j\ | \tag{7}$$

is set to govern the output of each joint actuator. In practice, violation of any of these constraints may cause considerable deviation from the desired planned motion.

III : The OLOCQS Algorithm

The algorithm proposed is termed the *On-Line Optimum Cubic-Quadratic Splines* or, the *OLOCQS* for short. This naming will be referred to hereafter. It is based on the previous work reported in [24, 25]. The joint-trajectories required for planning the manipulator motion are composed of successive polynomial segments between every two look-ahead points, hence providing the continuity required. A sample joint-trajectory is shown in figure (1), with *n* segments and *n* look-ahead points (excluding the starting point). The manipulator tip is assumed initially to be at point #1, with zero initial velocity and acceleration. The first look-ahead point (i.e. point #2) is then detected by the on-board sensory equipment, giving the position and orientation of the robot end-effector, and a trajectory segment is constructed between both points. However, once the manipulator starts to traverse the first segment, a second look-ahead point (point #3) is detected, and planning is resumed for the second segment of the trajectory. Hence, providing that the planning procedure for the second segment is completed before the manipulator reaches the end of the first segment, continuity of motion is guaranteed. Defining the following

$$t_{travel}^{i-1} \equiv \textit{the travelling time between points } i-1 \text{ and } i \ ,$$

$$t_{exec}^{i} \equiv \textit{the execution–time for planning a movement between points } i \text{ and } i+1 \ ,$$

then the following condition

$$t_{exec}^{i} \leq t_{travel}^{i-1} \tag{8}$$

is necessary and sufficient to ensure accurate tracking. However, if the planning time exceeds that of the traversal, the manipulator would be in an indeterment state, and effectively out of control. Therefore, the main task on hand is to accomplish as shorter a planning cycle as possible.

These trajectory segments are usually constructed by employing a set of spline functions in order to provide a continuous smooth motion. Cubic splines are the most applicable [13, 2] due to their attractive characteristics [23, 6]. This form of polynomial splines has been utilized in a previous attempt to design an on-line planner [5]. Another earlier attempt was made [4] employing a trapizoidal acceleration profile [17]. However, both of these attempts only constructed a single feasable trajectory, for which the minimum-time criterion was not accounted for.

III.1: *General Interpretation :*

The time-minimization problem addressed by the presented algorithm depends on fitting a *combined spline* of both *cubic* and *quadratic* polynomials for each segment of the trajectory, and further varying the produced trajectory in an attempt to optimise the travelling time. The search technique involved will be discussed in section III.4.3.

In order to perform the required optimisation procedure, the path of figure (1) is subdivided as shown in figure (2), providing a total of 4 via-points for each segment. This configuration would allow a cubic spline to be fitted for each 4-point segment, as shown in the following section.

III.2: *Splines formulation* :

The general theory of splines can be found in [6] while one application of cubic splines in the planning of robot trajectories is illustrated in [13].

The planning of a segment-trajectory is to be performed by 3 methods, according to the boundary conditions assumed for that specific segment, as follows:

(1) Planning the 1*st segment :* The initial values of position, velocity, and acceleration is to be specified at the start of the interval, and continuity of position and velocity is to be maintained at the end-point. The initial conditions for velocity and acceleration are usually taken to be zero.

(2) Planning the nth segment : Continuity of position and velocity must be guaranteed at the start point, and final conditions for position, velocity, and acceleration should be met at the end-point (i.e. *nth* look-ahead point). The latter are, again, assumed to be zero.

(3) Other intermediate segments : should account for the continuity of position and velocity at their start and end points.

Hence, considering all the continuity and boundary conditions required by each individual case, a system of l equations in l acceleration variables is constructed, taking into account values of position and time intervals provided, where l=3 for the 1*st* and *nth* segment, and l=4 otherwise. The time intervals are defined as

$$h_i = t_{i+1} - t_i \tag{9}$$

and are initially given the value

$$h_i = \frac{\theta_{i+1} - \theta_i}{\dot{\theta} limit_j} \tag{10}$$

for each joint segment. The system is represented as

$$\mathbf{A} \,.\, \mathbf{x} = \mathbf{b} \tag{11}$$

where, **x** is the required acceleration vector. The linear system of equation for each case is defined in the following :

* *for the* 1*st segment* :

$$\mathbf{A} = \begin{bmatrix} 3h_1+2h_2+\dfrac{h_1^2}{h_2} & h_2 & 0 \\ h_2-\dfrac{h_1^2}{h_2} & 2(h_3+h_3) & h_3 \\ 0 & h_3 & 2h_3 \end{bmatrix} \tag{12.1}$$

$$\mathbf{b} = \begin{bmatrix} 6\left[\left(\dfrac{\theta_3}{h_2}+\dfrac{\theta_1}{h_1}\right) - \left(\dfrac{1}{h_1}+\dfrac{1}{h_2}\right)\left(\theta_1+h_1\dot{\theta}_1+\dfrac{h_1^2\ddot{\theta}_1}{3}\right)\right] - h_1\ddot{\theta}_1 \\ 6\left[\left(\theta_1+h_1\dot{\theta}_1+\dfrac{h_1^2\ddot{\theta}_1}{3}\right)\dfrac{1}{h_2} + \dfrac{\theta_4}{h_3} - \left(\dfrac{1}{h_3}+\dfrac{1}{h_2}\right)\theta_3\right] \\ 6\left[\dot{\theta}_4 - \left(\dfrac{\theta_4-\theta_3}{h_3}\right)\right] \end{bmatrix} \tag{12.2}$$

$$\mathbf{x} = \begin{bmatrix} \ddot{\theta}_2 \\ \ddot{\theta}_3 \\ \ddot{\theta}_4 \end{bmatrix} \tag{12.3}$$

* *for the nth segment* :

$$\mathbf{A} = \begin{bmatrix} 2h_1 & h_1 & 0 \\ h_1 & 2(h_1+h_2) & h_2-\dfrac{h_3^2}{h_2} \\ 0 & h_2 & 3h_3+2h_2+\dfrac{h_3^2}{h_2} \end{bmatrix} \tag{13.1}$$

$$\mathbf{b} = \begin{bmatrix} 6\left[\left(\dfrac{\theta_2-\theta_1}{h_1}\right)-\dot{\theta}_1\right] \\ 6\left[\left(\theta_4-\dot{\theta}_4 h_3+\dfrac{h_3^2\ddot{\theta}_4}{3}\right)\dfrac{1}{h_2}-\left(\dfrac{1}{h_2}+\dfrac{1}{h_1}\right)\theta_2+\dfrac{\theta_1}{h_1}\right] \\ 6\left[-\left(\dfrac{1}{h_2}+\dfrac{1}{h_1}\right)\left(\theta_4-\dot{\theta}_4 h_3+\dfrac{h_3^2\ddot{\theta}_4}{3}\right)+\dfrac{\theta_4}{h_3}+\dfrac{\theta_2}{h_2}\right]-h_3\ddot{\theta}_4 \end{bmatrix} \tag{13.2}$$

$$\mathbf{x} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} \tag{13.3}$$

* _for other intermediate segment_ :

$$\mathbf{A} = \begin{bmatrix} 2h_1 & h_1 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 \\ 0 & 0 & h_3 & 2h_3 \end{bmatrix} \tag{14.1}$$

$$\mathbf{b} = \begin{bmatrix} 6\left[\left(\dfrac{\theta_2-\theta_1}{h_1}\right)-\dot{\theta}_1\right] \\ 6\left[\left(\dfrac{\theta_3-\theta_2}{h_2}\right)-\left(\dfrac{\theta_2-\theta_1}{h_1}\right)\right] \\ 6\left[\left(\dfrac{\theta_4-\theta_3}{h_3}\right)-\left(\dfrac{\theta_3-\theta_2}{h_2}\right)\right] \\ 6\left[\dot{\theta}_4-\left(\dfrac{\theta_4-\theta_3}{h_3}\right)\right] \end{bmatrix} \tag{14.2}$$

$$\mathbf{x} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \\ \ddot{\theta}_4 \end{bmatrix} \tag{14.3}$$

III.3: *The optimisation procedure* :

Each segment of the on-line trajectory constructed is similar to that shown in figure (3). Two procedures are applied to both intervals $h_{i-1}=[\ t_{i-1}\ ,\ t_i\ ]$ and $h_i=[\ t_i\ ,\ t_{i+1}\ ]$ to accomplish the minimum time criterion, namely: *time scaling* and *interval contraction.*

III.3.1: *Time scaling* :

For each of the indicated intervals, a scaling procedure is performed to achieve two purposes: (a) ensure the velocity, acceleration and jerk of both intervals do not exceed certain limits specified by the manipulator design for each of its joints, and (b) pull up the performance of the manipulator to a maximum, while taking (a) into account. Thus, the following is calculated,

$$K_1 = MAX\left[\underset{t\,\in\,[t_{i-1},t_i]}{MAX}\left[\frac{\dot{\theta}_{i-1}(t)}{\dot{\theta}limit_j}\right],\ \underset{t\,\in\,[t_i,t_{i+1}]}{MAX}\left[\frac{\dot{\theta}_i(t)}{\dot{\theta}limit_j}\right]\right] \tag{15}$$

$$K_2 = MAX\left[\frac{\ddot{\theta}_{i-1}(t_{i-1})}{\ddot{\theta}limit_j}\ ,\ \frac{\ddot{\theta}_i(t_i)}{\ddot{\theta}limit_j}\ ,\ \frac{\ddot{\theta}_i(t_{i+1})}{\ddot{\theta}limit_j}\right] \tag{16}$$

$$K_3 = MAX\left[\frac{\dddot{\theta}_{i-1}(t_{i-1})}{\dddot{\theta}limit_j}\ ,\ \frac{\dddot{\theta}_i(t_i)}{\dddot{\theta}limit_j}\right] \tag{17}$$

and,

$$K = MAX\left[K_1\ ,\ \sqrt{K_2}\ ,\ \sqrt{K_3}\right] \tag{18}$$

where, $\dot{\theta}$, $\ddot{\theta}$ and $\dddot{\theta}$ denote velocity, acceleration and jerk, respectively, and K denotes the *Scaling Factor*. Applying the factor K to the two intervals yields

$$h_{i-1} = K\ .\ h_{i-1} \tag{19}$$

$$h_i = K\ .\ h_i \tag{20}$$

$$\dot{\theta}_{i-1}(t_{i-1}) = \frac{1}{K}\ .\ \dot{\theta}_{i-1}(t_{i-1}) \tag{21}$$

$$\dot{\theta}_i(t_i) = \frac{1}{K}\ .\ \dot{\theta}_i(t_i) \tag{22}$$

$$\dot{\theta}_i(t_{i+1}) = \frac{1}{K}\ .\ \dot{\theta}_i(t_{i+1}) \tag{23}$$

$$\ddot{\theta}_{i-1}(t_{i-1}) = \frac{1}{K^2} \cdot \ddot{\theta}_{i-1}(t_{i-1}) \tag{24}$$

$$\ddot{\theta}_{i}(t_{i}) = \frac{1}{K^2} \cdot \ddot{\theta}_{i}(t_{i}) \tag{25}$$

$$\ddot{\theta}_{i}(t_{i+1}) = \frac{1}{K^2} \cdot \ddot{\theta}_{i}(t_{i+1}) \tag{26}$$

$$\dddot{\theta}_{i-1}(t_{i-1}) = \frac{1}{K^3} \cdot \dddot{\theta}_{i-1}(t_{i-1}) \tag{27}$$

$$\dddot{\theta}_{i}(t_{i}) = \frac{1}{K^3} \cdot \dddot{\theta}_{i}(t_{i}) \tag{28}$$

One exception is made in the case of the 1*st* and *nth* segments, where the other 3rd interval ,h_{i-2}, is included in the scaling procedure, and its parameters are checked as in *eqns*.(13–15).

III.3.2: *Interval contraction* :

For each of the intermediate segments, the two designated intervals (i.e. h_{i-1} and h_i), the location of the maximum velocity is found, and the corresponding position cubic function is truncated accordingly. Thus, for a time interval $[t_i,t_{i+1}]$, the maximum velocity could be at t_i , t_{i+1} or $t \in [t_i,t_{i+1}]$, which would cause the corresponding time interval to vanish, remain as it is, or be truncated at some value, respectively, in accordance with the standard properties of a cubic equation. However, since the initial and final conditions associated with the 1*st* and *nth* segments may be affected by such a procedure, only intervals h_i and h_{i-1} are processed in these two segments, respectively. The resulting time of each joint-segment is denoted $t^j_{cub[1]}$ and $t^j_{cub[2]}$, as it covers two cubic polynomials. In the case of the two end segments, a third interval, $t^j_{cub[3]}$ is also present.

III.4: *Quadratic Interpolation* :

The two procedures presented in section III.3 are applied to each joint-segment independently. Thus, maximum performance is expected to be achieved for all joints. It should be noted, however, that the optimisation procedure yields a time history of only part of the segment processed, and produces different travelling times for each joint of the robot. Hence, the remaining part of the segment should be constructed so as to provide the remaining time-history, while setting an equal travelling time for all joints. The rest of the segment is constructed by an approximated quadratic spline.

However, a *time compensation* process must be conducted first to achieve equality in travelling time.

III.4.1: *Time compensation* :

The following procedure is applied to all segments, except the 1*st*, since it contains no quadratic intervals. Hence, since the second derivative of a quadratic is a constant, each quadratic interval is initially set so as to get the maximum allowable acceleration, $\ddot{\theta}limit_j$,

$$h^{i,j}_{quad} = \left| \frac{\dot{\theta}_2(t_{i-1}) - \dot{\theta}_1(t_{i-2})}{\ddot{\theta}limit_j} \right| \tag{29}$$

This scheme is shown pictorically in figure(4) for segment i of a 6 joints manipulator (i.e. N=6), where *solid* lines represent cubic equations and *dashed* lines quadratics. Obviously, since each joint is being treated individually, one (or more) joint(*s*) is expected to reach the end point faster than the others (e.g. joint #4 in figure(4)). Thus, it is appropriate to add some additional time

$$t^{i,j}_{add} \quad , j=1,2,\ldots,N \tag{30}$$

for each lagging joint to make its total travelling time equal to that of the leading joint. In figure(4), $t^{i,j}_{add}$ is added for $j \in \{1,3,4,5,6\}$ to achieve equal travelling times for all six joints. Hence, the value of $h^{i,j}_{quad}$ in each joint-segments is adjusted so as to satisfy (a) the maximum allowable acceleration (*eqn*.29), and (b) equality of total segment-time over all joints. The value of the added time would be

$$t^{i,j}_{add} = \underset{j=1,2,..,N}{MAX} \left[h^{i-1,j}_{cub[2]} + h^{i,j}_{quad} + h^{i,j}_{cub[1]} \right] - \left[h^{i-1,j}_{cub[2]} + h^{i,j}_{quad} + h^{i,j}_{cub[1]} \right] \tag{31}$$

where, $h^{i-1,j}_{cub[2]}$ is the 2*nd* cubic interval of the previous planned segment, i−1. Thus, the values of the quadratic intervals are updated as follows

$$h^{i,j}_{quad} = h^{i,j}_{quad} + t^{i,j}_{add} \tag{32}$$

Once this procedure is applied to the example of figure (4), equal time segments are expected between all pairs of the look-ahead points, as shown in figure (5).

III.4.2: *Approximating the quadratic* :

Now, to fit in a quadratic equation between any two positions $\theta(t_i)$ and $\theta(t_{i+1})$, and velocities $\dot{\theta}(t_i)$ and $\dot{\theta}(t_{i+1})$, the corresponding time interval must be given as

$$h_{quad}^{i,j*} = 2\left[\frac{\theta(t_{i+1}) - \theta(t_i)}{\dot{\theta}(t_i) + \dot{\theta}(t_{i+1})}\right] \tag{33}$$

However, since $h_{quad}^{i,j}$ cannot be guaranteed to equal $h_{quad}^{i,j*}$, an approximation for the quadratic polynomial must be made. The derivation of (eqn.29) can be found in *Appendix* (*A*).

The approximation is accomplished by utilizing an equal interval, large-number-of-segments cubic spline. The time interval $h_{quad}^{i,j}$ is divided into l equal sub-intervals

$$h_{div} = \frac{h_{quad}^{i,j}}{l} \tag{34}$$

Since the velocity during interval $h_{quad}^{i,j}$ is a first order polynomial (i.e. a linear variant with time), the corresponding velocity at the start of each h_{div} interval is calculated as

$$\dot{\theta}_w = \dot{\theta}(t_i) + w \, . \left[\frac{\dot{\theta}(t_{i+1}) - \dot{\theta}(t_i)}{l}\right] \quad , \qquad w = 1,2,\ldots,l-1 \tag{35}$$

Thus a system of $l-2$ equations in $l-2$ *position* variables is constructed, taking into account values of h_{div} and $\dot{\theta}_w$ calculated. The system is represented as in *eqn*.(11), where

$$\mathbf{A} = \begin{bmatrix} \frac{-6}{h_{div}^2} & \frac{3}{h_{div}^2} & 0 & . & . & . & . & 0 \\ 0 & \frac{3}{h_{div}^2} & \frac{-6}{h_{div}^2} & \frac{3}{h_{div}^2} & . & . & . & . \\ . & 0 & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & 0 \\ 0 & . & . & . & . & 0 & \frac{3}{h_{div}^2} & \frac{-6}{h_{div}^2} \end{bmatrix} \tag{36}$$

and,

$$\mathbf{B} = \begin{bmatrix} \dfrac{-3\theta_1}{h_{div}^2} \\ \dfrac{-\dot{\theta}_1 - \dot{\theta}_2 + 2\,\dot{\theta}_3}{h_{div}} \\ \cdot \\ \cdot \\ \cdot \\ \dfrac{-3\theta_l}{h_{div}^2} \end{bmatrix} \tag{37}$$

and $\mathbf{x}$ is the required position vector,

$$\mathbf{x} = \begin{bmatrix} \theta_2 \\ \theta_3 \\ \cdot \\ \cdot \\ \theta_{l-2} \\ \theta_{l-1} \end{bmatrix} \tag{38}$$

It was found that as $l \rightarrow \infty$, then $h_{div} \rightarrow 0$, and the cubic polynomials concerned tend to form a quadratic. However, the value of l should be properly chosen for each approximation, to assure minimum discontinuity among sub-intervals, and to avoid unnecessary computational burden. The linear system described has been derived utilizing the *Hermit interpolation polynomial.* Detailed derivations are included in *Appendix* (*B*).

III.5: *The search technique* :

The planning procedure described in previous sections (III.2-III.4) has been concerned with only one single segment, which may not be necessarily the best choice. Thus, to obtain optimality in the time criterion, a *search* procedure must be employed. This is made possible by varying the positions of all via-points in a segment except for the one representing a *look–ahead* point, by a certain value θ_{var}. Furthermore, the procedures of sections III.2,III.3 and III.4.1 are applied, yielding yet another optimised segment. In order to compare the two resultant minimum-time segments, the following is defined according to the interpretation of figure (3) as

$$t^{orig} = h_i^{orig} + h_{i-1}^{orig} + \frac{\dot{\theta}^{orig}(t_{i-1}) - \dot{\theta}^{orig}(t_{i-2})}{\ddot{\theta}limit_j} \tag{39}$$

and,

$$t^{new} = h_i^{new} + h_{i-1}^{new} + \frac{\dot{\theta}^{new}(t_{i-1}) - \dot{\theta}^{orig}(t_{i-2})}{\ddot{\theta}limit_j} \tag{40}$$

where superscripts *orig* and *new* denotes *original* and *new* segments, respectively. Hence, for a new segment to be accepted as a *possible optimum*, the following condition must be satisfied

$$t^{new} < t^{orig} \tag{41}$$

It is noted from *eqns*.(39,40) that the initial velocity condition for both segments is the same (*i.e.* $\dot{\theta}^{orig}(t_{i-2})$). An exception is once again made for segment #1, where the following condition is imposed

$$h_i^{new} + h_{i-1}^{new} + h_{i-2}^{new} < h_i^{orig} + h_{i-1}^{orig} + h_{i-2}^{orig} \tag{42}$$

This process is repeated with a *negative* added value to the via-points, $-\theta_{var}$, and a yet newer segment is constructed, optimised, and chosen to be the minimum-time one if the condition of *eqn*.(41) (*or* (42)) is satisfied. At this stage, it would be appropriate to decide whether incrementing the positions or decrementing them produces greater minimization.

At another step in the search procedure, the value of θ_{var} is to be increased to $2\theta_{var}$, and the value would be added to (or subtracted from) the via-points, depending on the decision made in the previous step. An alternate segment is constructed and processed in a similar manner. The situation so far is illustrated by figure (6) for a single segment, and by figure (7) for the entire trajectory. This task is to be continued by adding

$$\theta_{add}^{u} = u \,.\, \theta_{var} \qquad , \; u = \pm\, 3,4,... \tag{43}$$

until no more minimum-time segments could be exchanged. However, it should be emphasized that the value of θ_{add}^{u} must not exceed the position bounds imposed by (*eqn*.3). If such a situation occurs, θ_{add}^{u} should be set to the corresponding bound value.

The initial task of the search process is to vary the value of velocity for each two-cubic part of the segment, leading to a smaller scaling factor, *K*, and therefore less time. The value of θ_{var} is chosen to be relatively large at the start, (e.g. 10 *degrees*), which was found to give better results than a smaller value. The search is stopped once the total segment time for a path with an incremental value θ_{add}^{u+1} is larger or equal to that with a value of θ_{add}^{u}.

Nevertheless, since a large incremental value of θ_{var} was used, larger minimization could be achieved considering a value $\theta_{add}^{u^*}$, where

$$\theta_{add}^{u} \leq \theta_{add}^{u^*} < \theta_{add}^{u+1} \tag{44}$$

Thus, a new search phase is initiated, with a start position of $\theta_{add}^{u^*}$ and an incremental value of

$$\theta_{add}^{u} = u \, . \, \frac{\theta_{div}}{10} \quad , \; u = \pm\, 1,2,3,... \tag{45}$$

where a new smaller travelling time would be achieved. A difference tolerance is assumed, δ, which represents the difference in the total travelling time between one phase of the search and the proceeding phase. If the value of δ was not met by the first two search phases described, then a third phase is to be initiated with

$$\theta_{add}^{u} = u \, . \, \frac{\theta_{div}}{100} \quad , \; u = \pm\, 1,2,3,... \tag{46}$$

and so forth until the value of δ is met.

Once the search is completed, the time compensation process of section III.3 is applied to the minimum-time segments obtained for all joints.

III.6: *Dynamic considerations* :

The limits on the trajectory performance considered in *eqns.*(4–6) are constant approximations that were deduced for simplicity. However, since the precence of coupling between the manipulator joints is an inherent fact, such an approximation is not totaly reliable [11]. Hence, a time scaling algorithm is to be employed, as described in [11, 14]. The equations of motion for a manipulator [18, 8] is defined as:

$$\tau_j = \sum_{q=1}^{N} D_{jq}\ddot{\theta}_q \; + \; \sum_{q=1}^{N}\sum_{m=1}^{N} h_{jqm}\, \dot{\theta}_q\, \dot{\theta}_m \; + \; c_j \qquad , \; j = 1,2,...,N \tag{47}$$

or, alternatively, in matrix form as

$$\boldsymbol{\tau}(t) = \mathbf{D}(\boldsymbol{\theta}(t))\; \ddot{\boldsymbol{\theta}}(t) \; + \; \mathbf{h}(\boldsymbol{\theta}(t),\dot{\boldsymbol{\theta}}(t)) \; + \; \mathbf{c}(\boldsymbol{\theta}(t)) \tag{48}$$

where,

$\boldsymbol{\tau}(t) = [\; \tau_1(t),\, \tau_2(t),\; \cdots \;, \tau_N(t)]^T = N\times 1$ *generalized torque vector* ,

$\boldsymbol{\theta}(t) = [\; \theta_1(t),\, \theta_2(t),\; \cdots \;, \theta_N(t)]^T = N\times 1$ *vector of joint variables* ,

$$\dot{\boldsymbol{\theta}}(t) = [\ \dot{\theta}_1(t), \dot{\theta}_2(t), \ \cdots\ , \dot{\theta}_N(t)]^T = N{\times}1 \ \textit{vector of joint velocity}\ ,$$

$$\ddot{\boldsymbol{\theta}}(t) = [\ \ddot{\theta}_1(t), \ddot{\theta}_2(t), \ \cdots\ , \ddot{\theta}_N(t)]^T = N{\times}1 \ \textit{vector of joint acceleration}\ ,$$

$$\mathbf{D}(\boldsymbol{\theta}(t)) = N{\times}N \ \textit{inertial acceleration– related matrix}\ ,$$

$$\mathbf{h}(\boldsymbol{\theta}(t),\dot{\boldsymbol{\theta}}(t)) = N{\times}1 \ \textit{nonlinear coriolis} \text{ and } \textit{centrifugal force vector}\ , \text{ and}$$

$$\mathbf{c}(\boldsymbol{\theta}(t)) = N{\times}1 \ \textit{gravity force vector}\ .$$

Thus, introducing a scaling factor, K^*, the new torque values would be

$$\tau_j^{new} = \frac{1}{(K^*)^2}\left[\tau_j - c_j\right] + c_j \tag{49}$$

or,

$$\tau_j^{new} = \sum_{q=1}^{N} D_{jq}\frac{\ddot{\theta}_q}{(K^*)^2} + \sum_{q=1}^{N}\sum_{m=1}^{N} h_{jqm}\frac{\dot{\theta}_q\,\dot{\theta}_m}{(K^*)^2} + c_j \tag{50}$$

The value of K^* can be found by considering the limit on the torques imposed by *eqn.*(7), then

$$K^* = \underset{j=1}{\overset{N}{MIN}}\left[\sqrt{\frac{\tau_j - c_j}{|\ \tau limit_j\ | - c_j}}\right] \tag{51}$$

The recursive Lagrangian formulation of the equations of motion were chosen for this application, as described in [10].

IV: The Computational Complexity

In this section, an attempt will be made to classify the computational complexities associated with each procedure of the *OLOCQS* algorithm. These processes are summarized in the following, along with their required execution time, ∇_*, for a single segment (i.e. one look-ahead point);

Process #1 : Transformation of a single point from the cartesian space to the configuration space (inverse kinematics algorithm); (∇_{IK}).

Process #2 : Splining and optimisation procedures, composing the search method, and including

a : three spline-fit (Gauss-Elimination) and optimisation procedures; [$3\ (\nabla^j_{spline} + \nabla^j_{opt})$].

b : ($r-1$) spline-fit and optimisation procedures, including phase #1 of the search; [$(r-1)\ (\nabla^j_{spline} + \nabla^j_{opt})$].

c : ($r(p-1)$) spline-fit and optimisation procedures, composing other search phases; [$r(p-1)\ (\nabla^j_{spline} + \nabla^j_{opt})$].

where p is the number of search phases and r is the number of the constructed segments of each. Thus, total process time would be $\nabla^j_{search} = (p.r+2)\ (\nabla^j_{spline} + \nabla^j_{opt})$.

Process #3 : Quadratic approximation; (∇^j_{quad}).

Process #4 : Dynamics considerations, (inverse dynamics algorithm; (∇_{ID}).

Hence, the total execution time required by the controller for a single segment i is

$$\begin{aligned} t^{i,j}_{exec} &= \nabla_{IK} + \nabla^j_{search} + \nabla^j_{quad} + \nabla_{ID} \\ &= \nabla_{IK} + (p.r+2)\ (\nabla^j_{spline} + \nabla^j_{opt}) + \nabla^j_{quad} + \nabla_{ID} \qquad (52) \end{aligned}$$

or, in global terms,

$$t^i_{exec} = \nabla_{IK} + \sum_{j=1}^{N} \left[\nabla^j_{search} + \nabla^j_{quad} \right] + \nabla_{ID} \qquad (53)$$

Therefore, if the condition of *eqn.*(8) is to be met, all the processes which require t^i_{exec} time should be accomplished while the manipulator is traversing the previous look-ahead segment, $i-1$. The travelling time of the 1*st* segment is defined as

$$t^{1,j}_{travel} = h^{1,j}_{cub[1]} + h^{1,j}_{cub[2]} + h^{1,j}_{cub[3]} \qquad (54)$$

or,

$$t^1_{travel} = \underset{j=1}{\overset{N}{MIN}}\ (\ t^{1,j}_{travel}\ ) \qquad (55)$$

We further define

$$h^{1,j}_{remain} = t^{1,j}_{travel} - t^{1}_{travel} \tag{56}$$

Also, for other segments $i-1$, i=3,4,...,n+1,

$$t^{i-1,j}_{travel} = h^{i-1,j}_{quad} + h^{i-1,j}_{cub[1]} + h^{i-1,j}_{cub[2]} + h^{i-2,j}_{remain} \tag{57}$$

or,

$$t^{i-1}_{travel} = \underset{j=1}{\overset{N}{MIN}} \left(t^{i-1,j}_{travel} \right) \tag{58}$$

and,

$$h^{i-1,j}_{remain} = t^{i-1,j}_{travel} - t^{i-1}_{travel} \tag{59}$$

The value of $h^{i-1,j}_{remain}$ is present because of the independent manipulation of each joint, which leads to a different travelling time. This has been made clear in figure (5).

It should be noted from *eqn.*(53) that the largest computational complexity is due to the value of ∇_{ID}, since this involves a multiple solution of the nonlinear equations of motion at some specific rate over the entire segment-time. Hence, an efficient and fast inverse dynamics algorithm is required to enhance the performance [26].

V: The Distributed Trajectory Generator

The computational complexities associated with the processes of the *OLOCQS* algorithm imposes a certain lower-limit upon the permissible minimum travelling time. It is desired to keep such a limit as small as possible in order to get a smaller control cycle leading to a better tracking performance. In our contribution to deal with this matter, the concepts of distributed processing have been adopted, leading to a considerable reduction in the execution time. The *OLOCQS* algorithm is readily structured for such an approach. However, since the travelling time of each of the look-ahead segments planned is more likely to differ, *dynamic assignment* of the involved procedures to the available processors has to be dealt with on-line. Hence, some sort of a *control unit* had to be constructed to perform the task of scheduling, which will be discussed in section V.2. In the following section, the distributed formulation is illustrated.

V.1: *The distributed formulation* :

Concurrency in the formulation is exploited at two distinct levels, as follows:

A. Global level : where each joint-segment is planned independently.

B. Local level : where the optimisation of possible joint-segments is performed by concurrent modules.

The proposed structure for the distributed formulation is illustrated in figure (8). It should be noted that each of the *Inverse Kinematics* and the *Inverse Dynamics* procedures is being treated as a sequential process, although a distributed form could be accommodated for, as will be discussed later.

It is proposed to map the *OLOCQS* algorithm onto M processors, supervised by the controller. Thus, for each joint of the manipulator, a number $L = \frac{M}{N}$ processors is available to perform the planning. Searching for the minimum-time segment is started by performing three *spline* and *optimisation* procedures, one for the original values of the via-points, and two others for the incremented and decremented alternatives. Upon the decision which side of the search is the best, a set of $(r-1)$ spline and optimisation procedures are initiated to decide on the minimum-time segment for this first phase of search. Depending on the accuracy of the minimization required (i.e the value of δ of section III.5), a set of $(p.r-r)$ similar procedures are involved. Once the time-optimum segment is found, a *quadratic approximation* procedure is used to produce the required segment.

The total execution-time required for the previous combination of tasks is

$$t_{exec}^{OLOCQS} = (p.r+2)\ (\ \nabla_{spline} + \nabla_{opt}\ ) + \nabla_{quad} \tag{60}$$

where each of the spline and optimisation procedures is proposed to be performed on a single processor. Adding the required time for the kinematics and dynamics computations, the total execution-time required is

$$t_{exec}^{i} = \nabla_{IK} + t_{exec}^{OLOCQS} + \nabla_{ID} \tag{61}$$

However, since the traversal-time t_{travel}^{i-1} cannot be guaranteed to accommodate for the required t_{exec}^{i}, a compromise should be sought.

V.2: <u>*The intelligent control unit (ICU)* :</u>

Since the travelling time of the previous segment (i.e. t_{travel}^{i-1}) may be less than that required to plan the present (i.e. t_{exec}^{i}), some of the processes covered by *eqn.*(61) may have to be ignored. Hence, an *intelligent control unit* should be created to supervise such a task. The *ICU* is to instruct the controller what number of the available processors is to be used. Such a decision is based heavily on the difference

$$\Delta t = t_{exec}^{i} - t_{travel}^{i-1} \tag{62}$$

If Δt is very large, it may be necessary to delete the dynamics evaluations, or computing them at a different rate, thus reducing the execution time by a significant amount. Otherwise, if the time difference is slight, a mere deletion of one or more phases (p) of the search may be adequate, or even a few segments (r) of each.

Although certain reductions in the execution time are acceptable, there is a certain lower bound that should be maintained,

$$t_{bound}^{i} = \nabla_{IK} + \nabla_{spline} + \nabla_{opt} \tag{63}$$

which represents merely the requirements for computing the inverse kinematics transformation of the look-ahead point in addition to constructing a single feasable segment.

We consider here the extreme case of having

$$t_{bound}^{i} > t_{travel}^{i-1} \tag{64}$$

which prevents the ICU from issuing the simplest solution offered by *eqn.*(63). Hence, the ICU should be programmed to check the value of the available travelling time a priori to executing its corresponding segment, and further rescaling the value of time, if necessary, to accommodate for the lower-bound imposed.

In summary, the ICU would act as a supervisor to the multiprocessor controller, activating periodically the appropriate number of processors. Its ultimate aim is to plan as accurate and optimum segment as possible for the manipulator to track, although a certain percentage of the accuracy and/or optimality sought could be waived away in certain critical situations, thus ensuring a continuous and predictable motion of the robot.

V.3: *The inverse dynamics and inverse kinematics computations* :

The solution for the inverse dynamics (ID) was considered as a single sequential unit in the formulations of sections V.1 and V.2. However, the ID process is to be computed at a certain rate defined by the robot control cycle [3]. Therefore, the required computational time is actually

$$\nabla_{ID} = t_{ID} \cdot \frac{t^{i}_{travel}}{t_{cycle}} \tag{65}$$

where,

$$t_{cycle} \equiv \textit{the period of the robot control cycle} \ ,$$

$$t_{ID} \equiv \textit{the time required to compute a single ID procedure} \ ,$$

Hence, reducing the value of t_{ID} would lead to a better efficiency. This can also be achieved by the application of parallel/pipelined structures, for which several contributions could be found in the related literature. One practical system is proposed in [26], which enables the computations at a speed of 26 microseconds per procedure, employing an MIMD array of transputers.

A similar argument could be made for the inverse kinematics algorithm, for which an efficient pipelined formulation had been developed in [12].

VI: Simulation Results

In this section, an attempt is made to prove the practicality and efficiency of the proposed trajectory generator by solving for the 6 degrees-of-freedom *PUMA* 560 robot manipulator as a case study. Simulation programs have been written in the *C programming language* and run on the *Sun Workstation.* A total of 7 look-ahead-points were chosen for each of the manipulator joints, as shown in Table (1). The constraints on the manipulator parameters are given in Table (2).

Table (1) : Joint Variables

Point#	Joint#					
	1	2	3	4	5	6
Start	10	15	45	5	10	6
1	60	25	180	20	30	40
2	75	30	200	60	-40	80
3	130	-45	120	110	-60	70
4	110	-55	15	20	10	-10
5	100	-70	-10	60	50	10
6	-10	-10	100	-100	-40	30
7	-50	10	50	-30	10	20

Table (2) : Manipulator Constraints

Limit	Joint#					
	1	2	3	4	5	6
Velocity ($deg.sec^{-1}$)	100	95	100	150	130	110
Acceleration ($deg.sec^{-2}$)	45	40	75	70	90	80
Jerk ($deg.sec^{-3}$)	60	60	55	70	75	70
Torque (*N.m.*)	97.6	186.4	89.4	24.2	20.1	21.3

Applying the *OLOCQS* algorithm, joint trajectories could be constructed in a look-ahead manner, as was described in previous sections. The minimum-time segments obtained are compared with the originals in Table (3), where the total travelling time was reduced from (34.117) down to (8.811) seconds.

Table (3) : Simulation Results

Trajectory Segment	Original Time *	Planned Minimum Time *
1	7.214	1.227
2	2.878	0.856
3	4.272	0.829
4	5.615	0.955
5	2.915	0.833
6	5.879	1.179
7	2.672	0.393
8	2.672	2.207

* values in seconds

The process of applying the optimization procedure and conducting the search technique is illustrated in Figure (9) for the 1*st* segment of joint #1 of the manipulator. In Figure (9a), several possible segments are constructed, where each is optimized independently, yielding different travelling time for each, as shown in Figure (9b). Finally, the optimum segment amid these is chosen, as in Figure (9c). All other look-ahead segments are processed in a similar fashion. Hence, the first *two* segments constructed are presented in Figure (10) for joint #1, while the complete minimum-time trajectories for all 6 joints of the *PUMA* are included in Figures (11-16). The initial and final conditions for velocity and acceleration at both start and end points are taken to be zero.

To elucidate the function of the *ICU*, values should be assigned for each of the execution times of *eqn.*(52) when run on the *Sun* microsystem with a floating point accelerator [20]. These values are arranged in Table (4).

Table (4) : Processes Execution Time

Execution Time	Value (*msec*)
∇^j_{spline}	12
∇^j_{opt}	65
t_{ID} *	67

* ∇_{ID} is as in *eqn.*(65)

However, as the execution time of the inverse kinematics algorithm, ∇_{IK}, is minute relative to the total completion time, it has been neglected in this simulation. Furthermore, taking into account the value of t_{ID} for the inverse dynamics algorithm, it was regerded to produce a significant overload in computations, particularly when a cycle-time of t_{cycle}= 28 milliseconds (i.e. control frequency of 36 *Hz*) is maintained for the *PUMA* [8].

As the construction of segment #1 is carried out while the manipulator is at rest, no limit is placed upon the planning (execution) time needed. However, the travelling time of the 1*st* segment, computed by *eqn.*(55) is to be taken as the planning time required for the next. the planning procedure of the 2*nd* segment is illustrated in Table (5) for all 6 joints of the manipulator. Consequently, the number of needed phases (*p*) of the search, along with the number of optimisation procedure (*r*) for each joint could be recognized, considering an accuracy tolerance of $\delta = 0.01$ for the optimisation. The minimum-time segment was found to be of duration t^2_{travel} = 1.422 seconds, and is to be planned within a time t^1_{exec} = 0.737 seconds. Although the planning time for most joints was within the required limit, the optimum planning time for joint #5 exceeded it (1.540 > 0.737), where a total of 5 search phases were needed to come in terms with the required tolerance, δ. Hence, the *ICU* was invoked to reduce the time by limiting the serches to 3 phases only, as shown in Table (5).

Table (5) : Planning of the 2nd Segment

Joint #	Search History							
	Optimality Requirements				*ICU* Instructions			
	p	r	Total	Planning Time *	p	r	Total	Planning Time *
1	3	3+2+2	7	0.539	3	3+2+2	7	0.539
2	3	3+2+2	7	0.539	3	3+2+2	7	0.539
3	3	3+2+1	6	0.462	3	3+2+1	6	0.462
4	3	3+2+2	7	0.539	3	3+2+2	7	0.539
5	5	3+2+7+7+1	20	1.540	3	3+2+4	9	0.693
6	3	3+2+2	7	0.539	3	3+2+2	7	0.539

* values in seconds

The fast algorithm for the inverse dynamics computations presented in [26] requires a time of 2.46 milliseconds per solution, employing a network of 4 transputers, which would have accommodated easily for the dynamic effects in this example.

The data used in this case study were obtained from [13,22] for comparison purposes, while the related *PUMA* data were extracted from [13,1].

VII: Conclusion

An algorithm has been developed for the minimum-time tracking of robot motion, where planning was performed in local basis for each joint trajectory. The proposed method has its main thrust for being able to generate the required trajectory in an on-line scheme. The introduction of the supervisory control unit design complements its performance capabilities, and ensure accurate and continuous motion to be possible. The computational burden caused by the global selection of different constraints for solving the planning problem was substantially reduced through a distributed realization of the formulation. The authors see the primary benefit of providing such an integrated system for certain applications where the precence of intelligent sensory-based robots is a must when complicated, varying and unstructured environment exists. Further research is currently being conducted, where the proposed algorithm is to be mapped on an array of T800 transputer machines, for which programming is performed in the parallel C language. Hence, the practicality of the algorithm would be shown through its implementation on an actual multiprocessor system.

References

[1]. ARMSTRONG, B., KHATIB, O., AND BURDICK, J., (1986). "The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm," *Proc. 1986 IEEE Int. Conf. on Robotics and Automation*, vol. 1, pp. 510-18.

[2]. BOLLINGER, J. AND DUFFIE, N., (1979). "Computer Algorithms for High Speed Continuous-Path Robot Manipulators," in *Annals of the CIRP*, vol. 28, pp. 391-95.

[3]. BRADY, J. M., HOLLERBACH, J. M., JOHNSON, T. L., LOZANO-PEREZ, T., AND MASON, M. T., (1982). *Robot Motion : Planning and Control*, MIT Press.

[4]. CASTAIN, R. H. AND PAUL, R. P., (1984). "An On-Line Dynamic Trajectory Generator," *Int. J. Robotics Research*, vol. 3, no. 1, pp. 68-72.

[5]. CHAND, S. AND DOTY, K. L., (1985). "On-Line Polynomial Trajectories for Robot Manipulators," *Int. J. of Robotics Research*, vol. 4, no. 2, pp. 38-48.

[6]. DE-BOOR, C., (1978). *A Practical Guide to Splines*, Springer-Verlag.

[7]. DICKINSON, M. AND MORRIS, A. S., (1988). "Co-ordinate Determination and Performance Analysis for Robot Manipulators and Guided Vehicles," *IEE Proceedings, Part-A*, vol. 135, no. 2, pp. 95-98.

[8]. FU, K. S., GONZALEZ, R. C., AND LEE, C. S. G., (1987). *Robotics : Control, Sensing, Vision and intelligence*, McGraw Hill.

[9]. HIRZINGER, G. AND DIETRICH, J., (1986). "Multisensory Robots and Sensorybased Path Generation," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 3, pp. 1992-2001.

[10]. HOLLERBACH, J. M., (1980). "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," *IEEE Trans. Syst., Man, Cyber.*, vol. SMC-10, pp. 730-36.

[11]. HOLLERBACH, J. M., (1984). "Dynamic Scaling of Manipulator Trajectories," *Trans. ASME, J. of Dyn. Syst., Meas. and Control* , vol. 106, pp. 102-06.

12]. LEE, C. S. G. AND CHANG, P. R., (1986). "A Maximum Pipelined CORDIC Architecture for Robot Inverse Kinematics Computation," Report TR-EE-86-5, Purdue University.

[13]. LIN, C. S., CHANG, P. R., AND LUH, J. Y. S., (1983). "Formulation and Optimization of Cubic Polynomial Joint Trajectories For Industrial Robots," *IEEE Trans. Automatic Control*, vol. AC-28, pp. 1066-74.

[14]. LIN, C. S. AND CHANG, P. R., (1985). "Approximate Optimum Paths of Robot Manipulators Under Realistic Physical Constraints," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 737-42.

[15]. LOZANO-PEREZ, T., (1987). "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE J. Robotics and Automation*, vol. RA-3, no. 3, pp. 224-38.

[16]. LUH, J. Y. S., WALKER, M. W., AND PAUL, R. P. C., (1980). "On-Line Computational Scheme for Mechanical Manipulators," *Trans. ASME, J. of Dyn. Syst., Meas. and Control*, vol. 102, pp. 69-76.

17]. PAUL, R. P., (1976). "Explaratory Research in Advanced Automation," SRI Project 4391, Chapter 4, 5th Report.

[18]. PAUL, R. P., (1981). *Robot Manipulators : Mathmatics, Programming and Control*, MIT Press.

19]. PORRILL, J., POLLARD, S. B., PRIDMORE, T. P., BOWEN, J. B., MAYHEW, J. E. W., AND FRISBY, J. P., (1988). "TINA: The Sheffield AIVRU Vision System," AIVRU memo #27, AI Vision

Research Unit, Sheffield University, United Kingdom.

20]. SUN,(1986), *Floating-Point Programmer's Guide for the Sun Workstation*, Sun Microsystems Inc..

[21]. SAHAR, G. AND HOLLERBACH, J. M., (1986). "Planning of Minimum-time Trajectories for Robot Arms," *Int. J. Robotics Research*, vol. 5, no. 3, pp. 90-100.

[22]. THOMPSON, S. E. AND PATEL, R. V., (1987). "Formulation of Joint Trajectories for Industrial Robots Using B-Splines," *IEEE Trans.* , vol. IE-34, pp. 192-199.

[23]. VANDERGRAFT, J. S., (1978). *Introduction to Numerical Computations*, Academic Press.

24]. ZALZALA, A. M. S. AND MORRIS, A. S., (1988). "An Optimum Trajectory Planner for Robot Manipulators in Joint-Space and Under Physical Constraints," Research Report #349, University of Sheffield, Department of Control Engineering.

[25]. ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "Optimum Trajectory Planning for Robot Manipulators," *To appear, IMA Conf. on Robotics: Applied Mathematics and Computational Aspects, 12-14 July*, Loughborough University of Technology, United Kingdom.

26]. ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "A Distributed Pipelined Architecture of the Recursive Lagrangian Equations of Motion for Robot Manipulators with VLSI Implementation," Research Report #353, Department of Control Engineering, University of Sheffield, United Kingdom.

Appendix (A)
Quadratic Splines

Since $\theta(t)$ is a quadratic, then $\dot{\theta}(t)$ must be a linear function

$$\dot{\theta}(t) = \frac{\dot{\theta}_i}{h_i}(t_{i+1}-t) + \frac{\dot{\theta}_{i+1}}{h_i}(t-t_i) \qquad (A.1)$$

$$, i=1,2,\ldots,m$$

where,

$$h_i = t_{i+1} - t_i \qquad (A.2)$$

and m is the number of points.

Integrating $(A.1)$ yields

$$\dot{\theta}(t) = -\frac{\dot{\theta}_i}{2h_i}(t_{i+1}-t)^2 + \frac{\dot{\theta}_{i+1}}{2h_i}(t-t_i)^2 + C_1 \qquad (A.3)$$

Evaluating C_1 yields

$$\theta(t_i) = \theta_i = -\frac{\dot{\theta}_i h_i}{2} + C_1 \qquad (A.4)$$

$$\Rightarrow C_1 = \theta_i + \frac{\dot{\theta}_i h_i}{2} \qquad (A.5)$$

also,

$$\theta(t_{i+1}) = \theta_{i+1} = \frac{\dot{\theta}_{i+1} h_i}{2} + C_1 \qquad (A.6)$$

$$\Rightarrow C_1 = \theta_{i+1} - \frac{\dot{\theta}_{i+1} h_i}{2} \qquad (A.7)$$

Equating (A.5) and (A.7) gives

$$\theta_{i+1} = \theta_i + \frac{h_i}{2}(\dot{\theta}_i+\dot{\theta}_{i+1}) \qquad (A.8)$$

Appendix (B)

Velocity Approach to the Formulation of Cubic Splines

For an interval with only two points known, the *Hermit Interpolating Formula* can be expressed as

$$\theta(t) = \dot{\theta}_{k-1}\left[\frac{(t_k-t)^2(t-t_{k-1})}{h_k^2}\right] - \dot{\theta}_k\left[\frac{(t-t_{k-1})^2(t_k-t)}{h_k^2}\right] +$$

$$\theta_{k-1}\left[\frac{(t_k-t)^2[2(t-t_{k-1})+h_k]}{h_k^3}\right] + \theta_k\left[\frac{(t-t_{k-1})^2[2(t_k-t)+h_k]}{h_k^3}\right] \qquad \text{(B.1)}$$

where,

$$h_k = t_{k+1} - t_k \qquad \text{(B.2)}$$

then,

$$\ddot{\theta}(t_k) = \frac{2}{h_k}(\dot{\theta}_{k-1}+2\dot{\theta}_k) - 6\left[\frac{\theta_k-\theta_{k-1}}{h_k^2}\right] \qquad \text{(B.3)}$$

$$\ddot{\theta}(t_{k+1}) = \frac{2}{h_{k+1}}(\dot{\theta}_k+2\dot{\theta}_{k+1}) - 6\left[\frac{\theta_{k+1}-\theta_k}{h_{k+1}^2}\right] \qquad \text{(B.4)}$$

Equating (B.3) and (B.4) yields

$$\frac{2}{h_k}(\dot{\theta}_{k-1}+2\dot{\theta}_k) - 6\left[\frac{\theta_k-\theta_{k-1}}{h_k^2}\right] = \frac{2}{h_{k+1}}(\dot{\theta}_k+2\dot{\theta}_{k+1}) - 6\left[\frac{\theta_{k+1}-\theta_k}{h_{k+1}^2}\right] \qquad \text{(B.5)}$$

Setting equal values for all values of h's and simplifying gives

$$\frac{3}{h^2}\theta_{k-1} - \frac{6}{h^2}\theta_k + \frac{3}{h^2}\theta_{k+1} = \frac{1}{h}[-\dot{\theta}_{k-1}-\dot{\theta}_k+2\dot{\theta}_{k+1}] \qquad \text{(B.6)}$$

$$, \quad k=2,3,\ldots,l-1$$

where l is the number of points. Since the values of θ_1 and θ_l are known, then for k=2,

$$-\frac{6}{h^2}\theta_2 + \frac{3}{h^2}\theta_3 = \frac{1}{h}[-\dot{\theta}_1-\dot{\theta}_2+2\dot{\theta}_3] - \frac{3}{h^2}\theta_1 \qquad \text{(B.7)}$$

also for $k=l-1$,

$$\frac{3}{h^2}\theta_{l-2} - \frac{6}{h^2}\theta_{l-1} = \frac{1}{h}[-\dot{\theta}_{l-2}-\dot{\theta}_{l-1}+2\dot{\theta}_l] - \frac{3}{h^2}\theta_l \qquad \text{(B.8)}$$

Thus, a system of $l-2$ equations can be solved to obtain the required position values θ_k, $k=1,2,\ldots,l-1$.

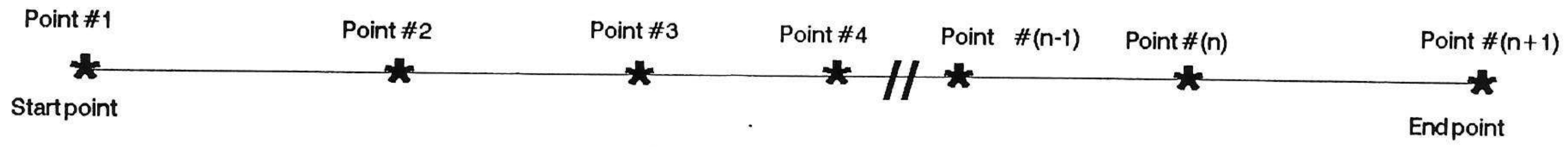


Figure (1)

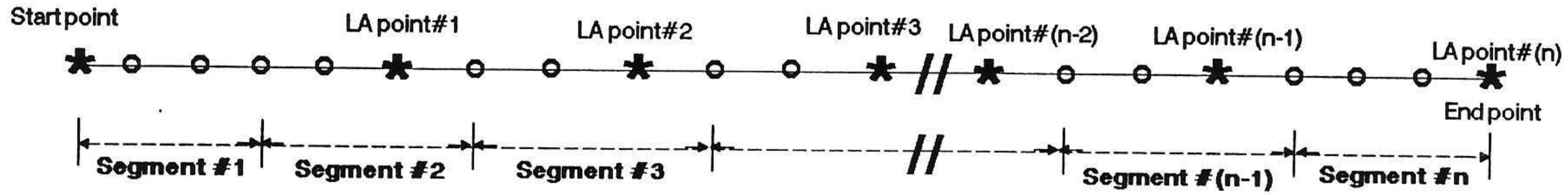


Figure (2)

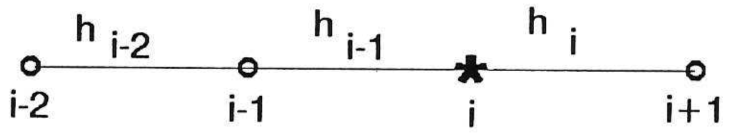


Figure (3)

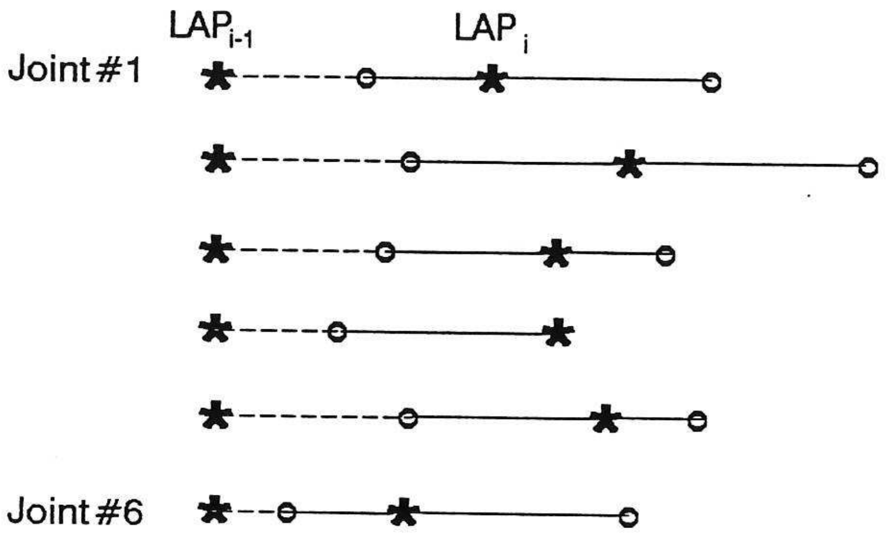


Figure (4)

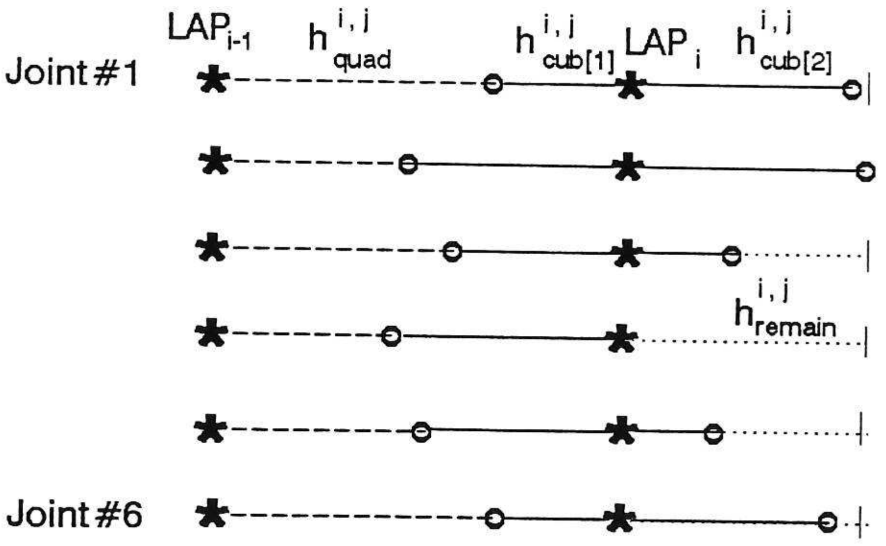


Figure (5)

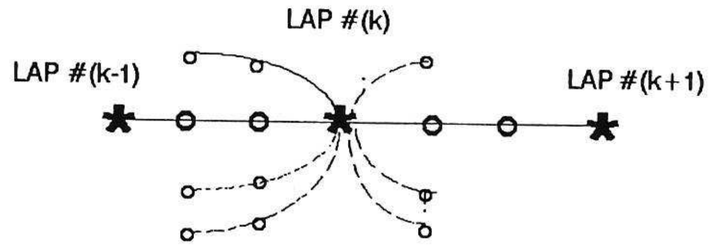


Figure (6)

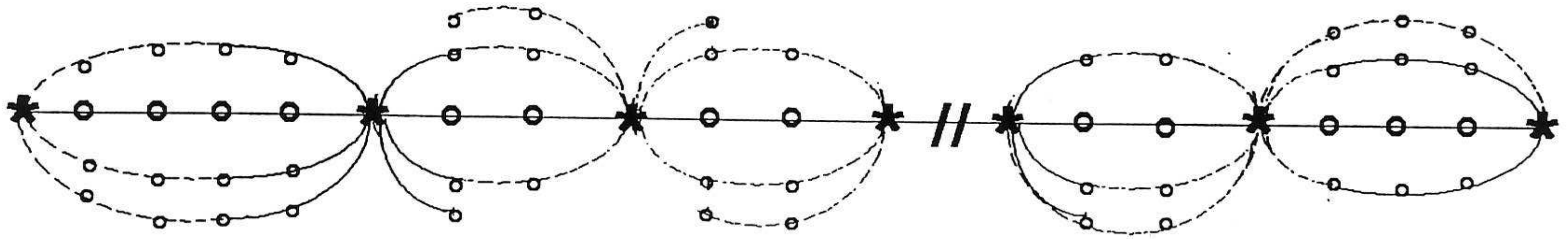


Figure (7)

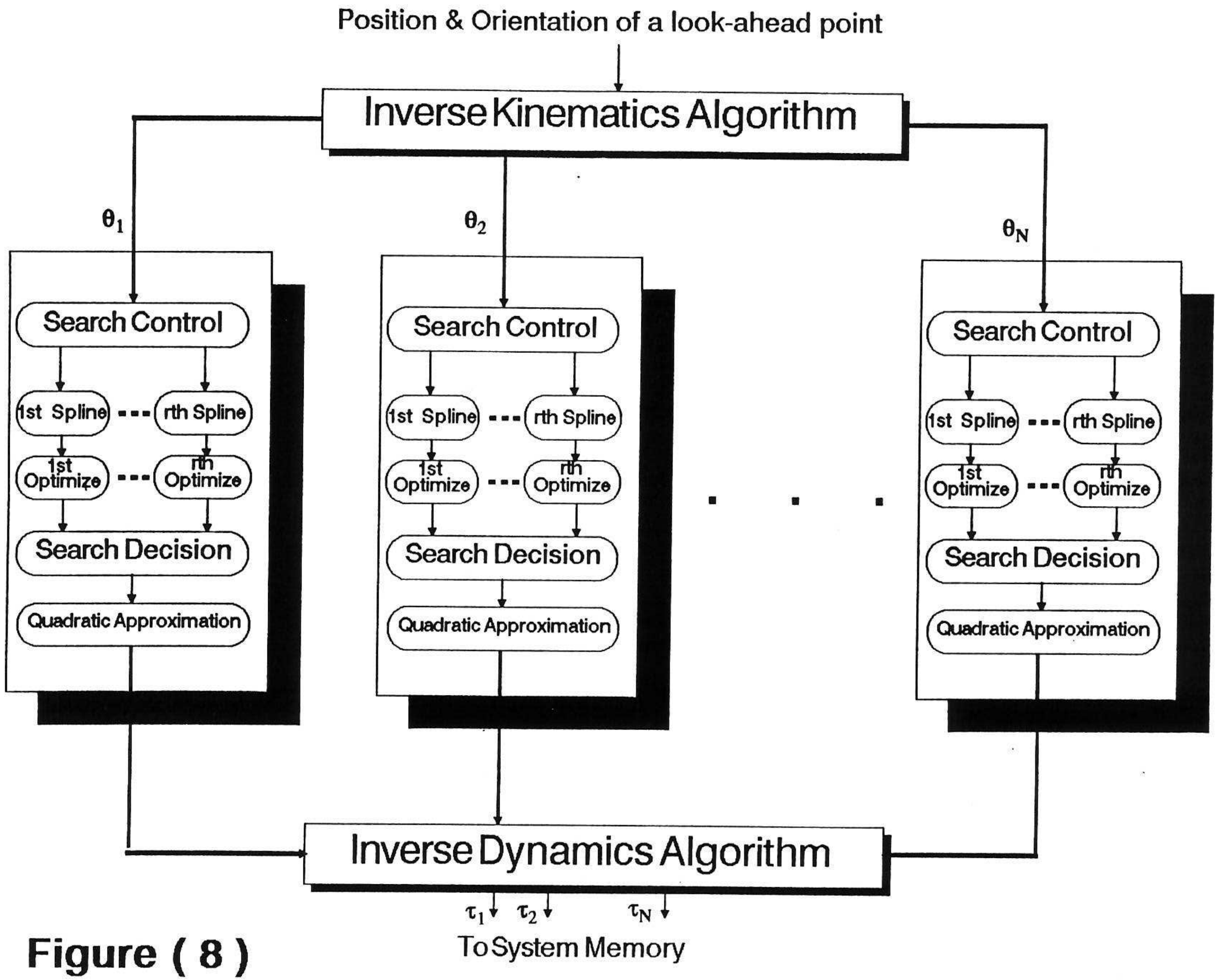

Position & Orientation of a look-ahead point
Inverse Kinematics Algorithm
θ_1
θ_2
θ_N
Search Control
1st Spline
rth Spline
1st Optimize
rth Optimize
Search Decision
Quadratic Approximation
Inverse Dynamics Algorithm
τ_1
τ_2
τ_N
To System Memory

Figure (8)

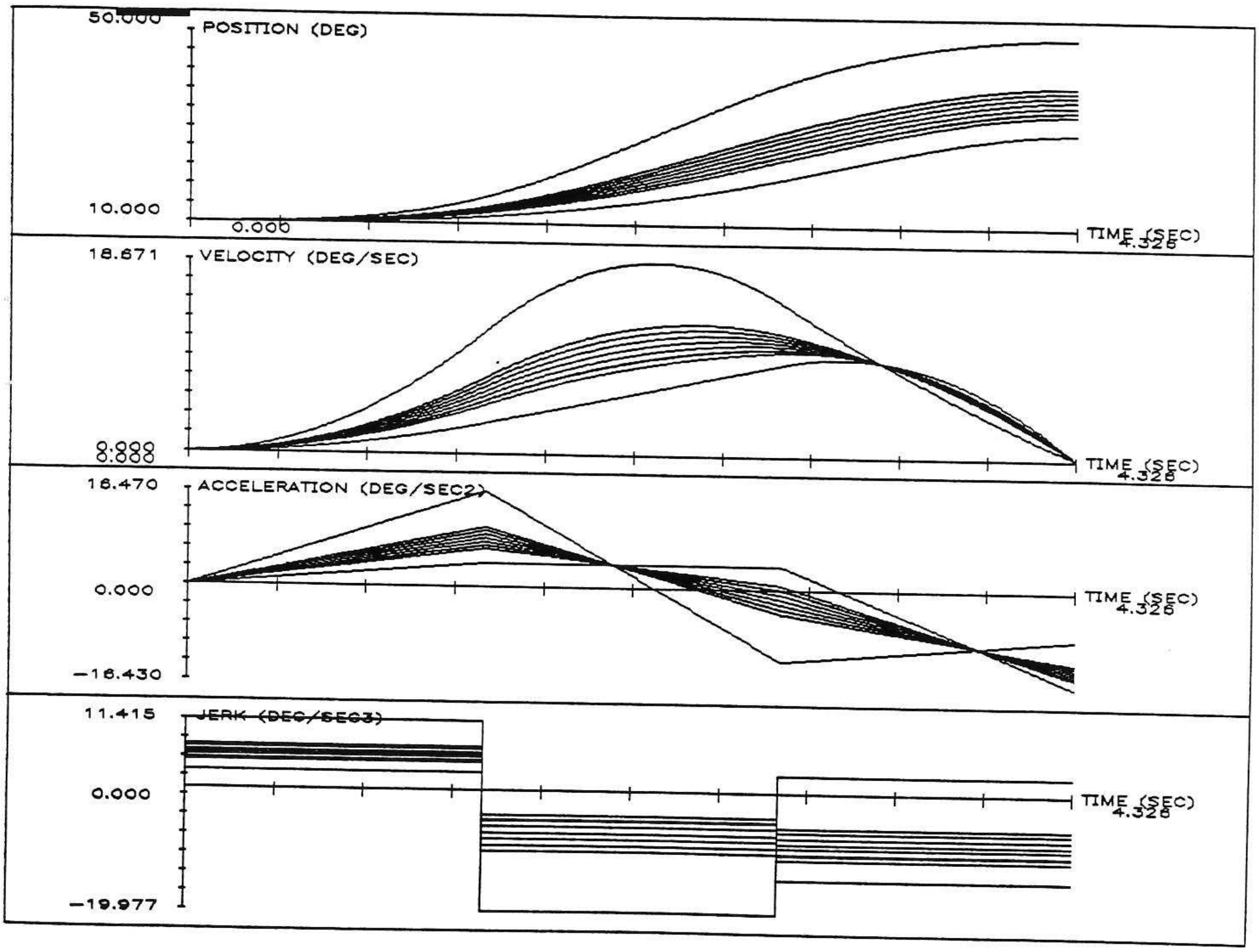

50.000
POSITION (DEG)
10.000
0.000
TIME (SEC)
4.328
18.671
VELOCITY (DEG/SEC)
0.000
0.000
TIME (SEC)
4.328
16.470
ACCELERATION (DEG/SEC2)
0.000
TIME (SEC)
4.328
-16.430
11.415
JERK (DEG/SEC3)
0.000
TIME (SEC)
4.328
-19.977

Figure 9a

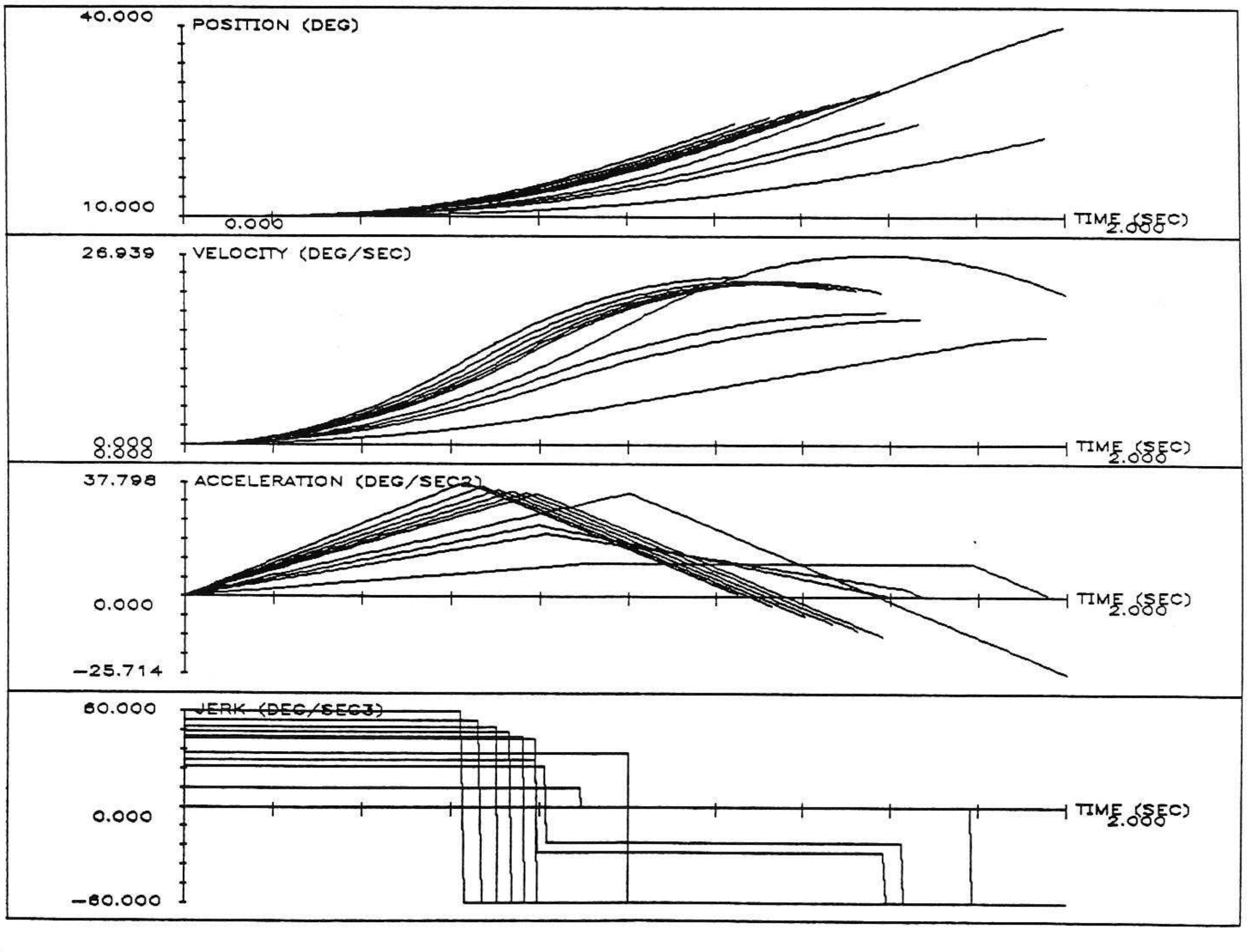

40.000
POSITION (DEG)
10.000
0.000
TIME (SEC)
2.000
26.939
VELOCITY (DEG/SEC)
0.000
TIME (SEC)
2.000
37.798
ACCELERATION (DEG/SEC2)
0.000
−25.714
TIME (SEC)
2.000
60.000
JERK (DEG/SEC3)
0.000
−60.000
TIME (SEC)
2.000

Figure 9b

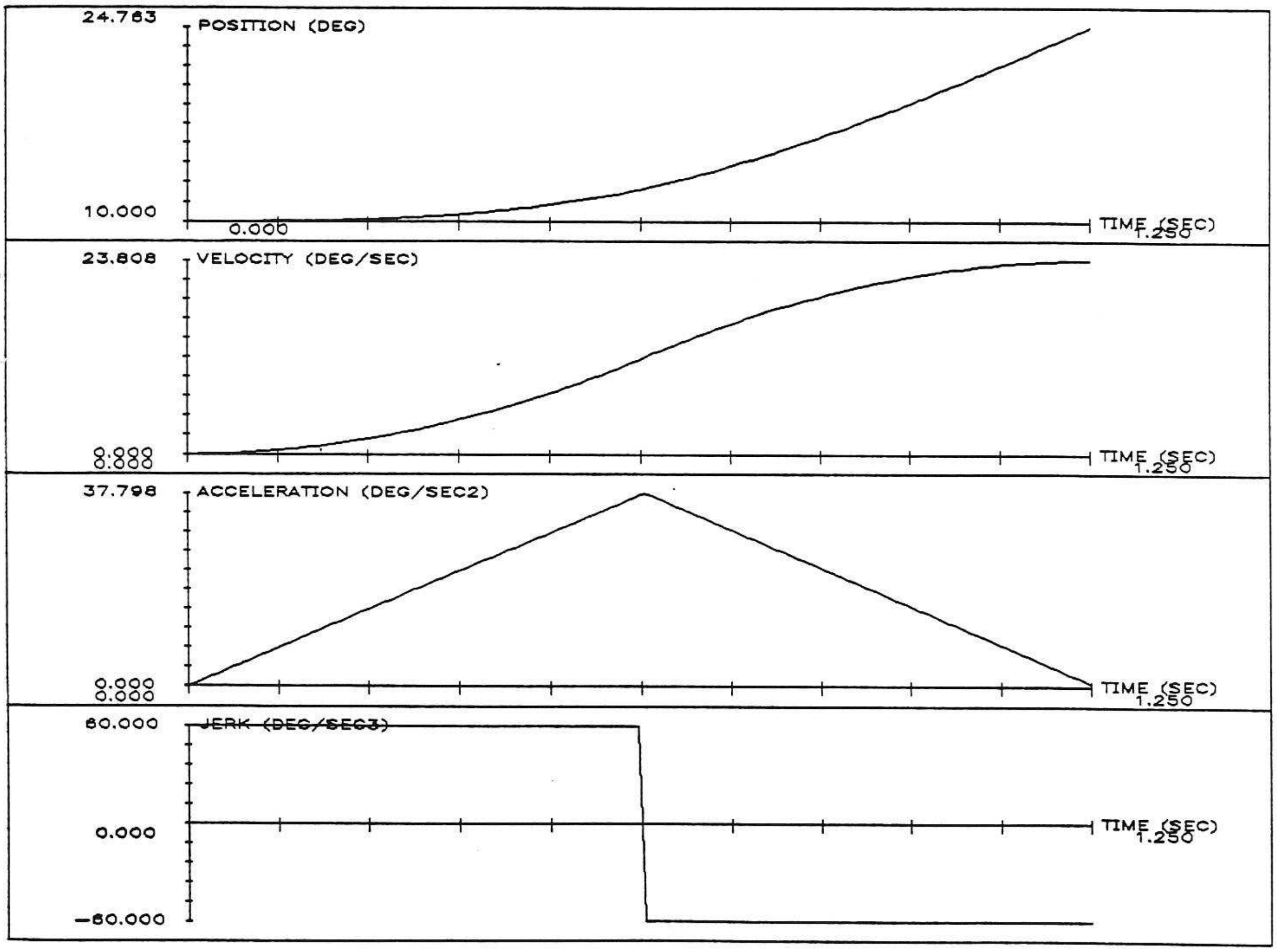
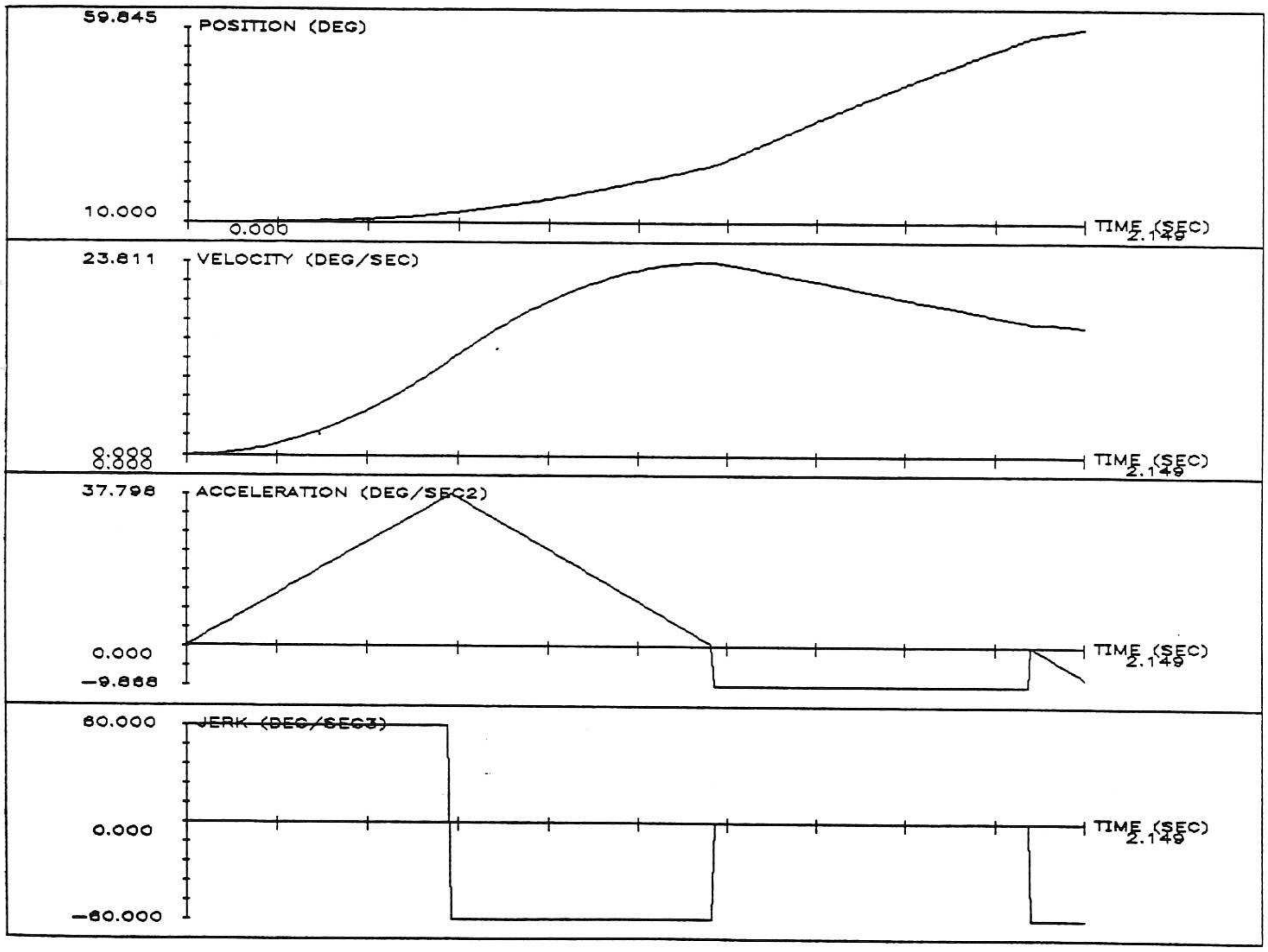

24.763
POSITION (DEG)
10.000
0.000
TIME (SEC)
1.250
23.808
VELOCITY (DEG/SEC)
0.000
0.000
TIME (SEC)
1.250
37.798
ACCELERATION (DEG/SEC2)
0.000
0.000
TIME (SEC)
1.250
60.000
JERK (DEG/SEC3)
0.000
TIME (SEC)
1.250
-60.000

Figure 9c



59.845
POSITION (DEG)
10.000
0.000
TIME (SEC)
2.149
23.811
VELOCITY (DEG/SEC)
0.000
TIME (SEC)
2.149
37.798
ACCELERATION (DEG/SEC2)
0.000
-9.868
TIME (SEC)
2.149
60.000
JERK (DEG/SEC3)
0.000
TIME (SEC)
2.149
-60.000

Figure 10

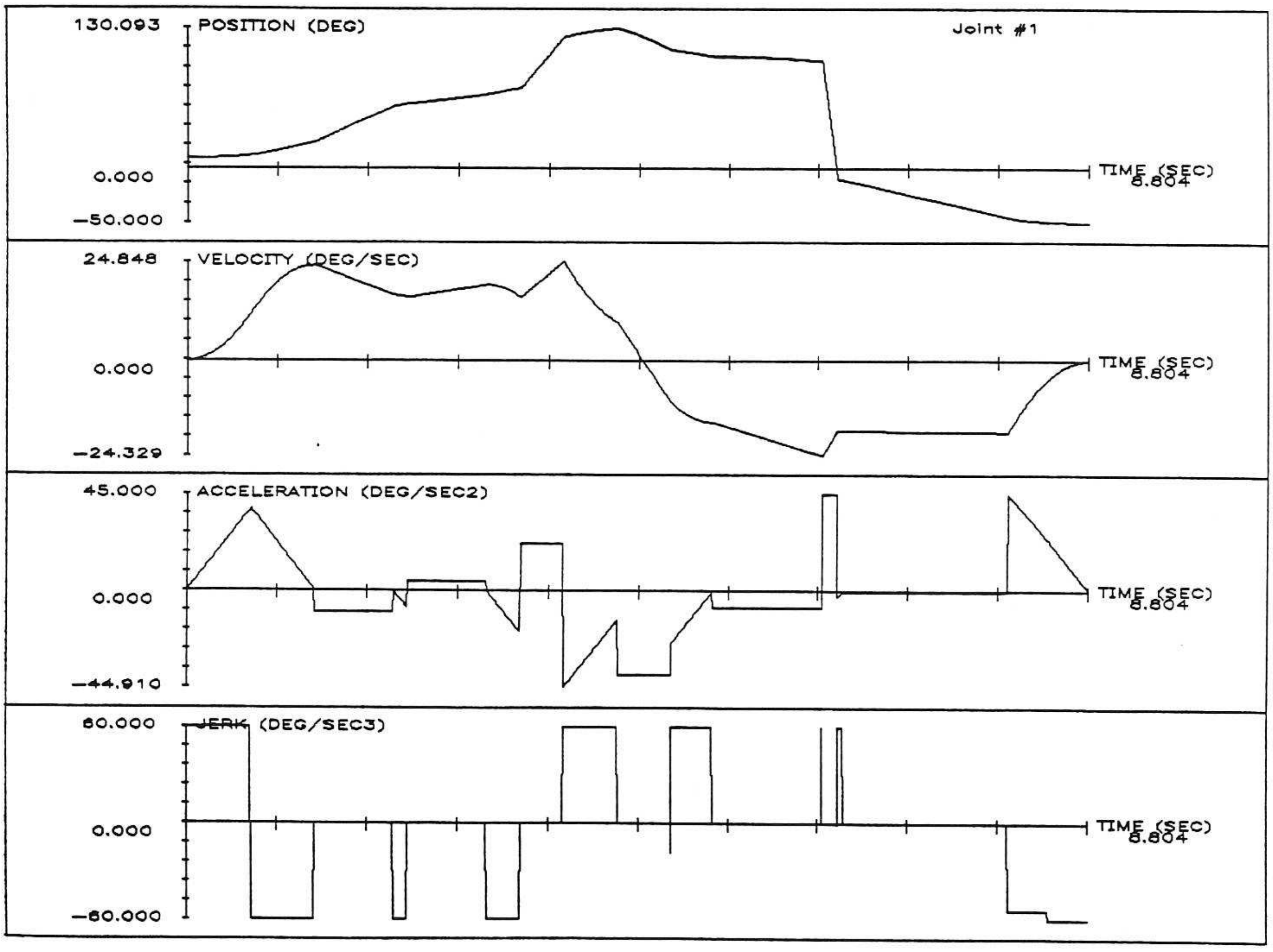

130.093
POSITION (DEG)
Joint #1
0.000
-50.000
TIME (SEC)
8.804
24.848
VELOCITY (DEG/SEC)
0.000
-24.329
TIME (SEC)
8.804
45.000
ACCELERATION (DEG/SEC2)
0.000
-44.910
TIME (SEC)
8.804
60.000
JERK (DEG/SEC3)
0.000
-60.000
TIME (SEC)
8.804

Figure 11

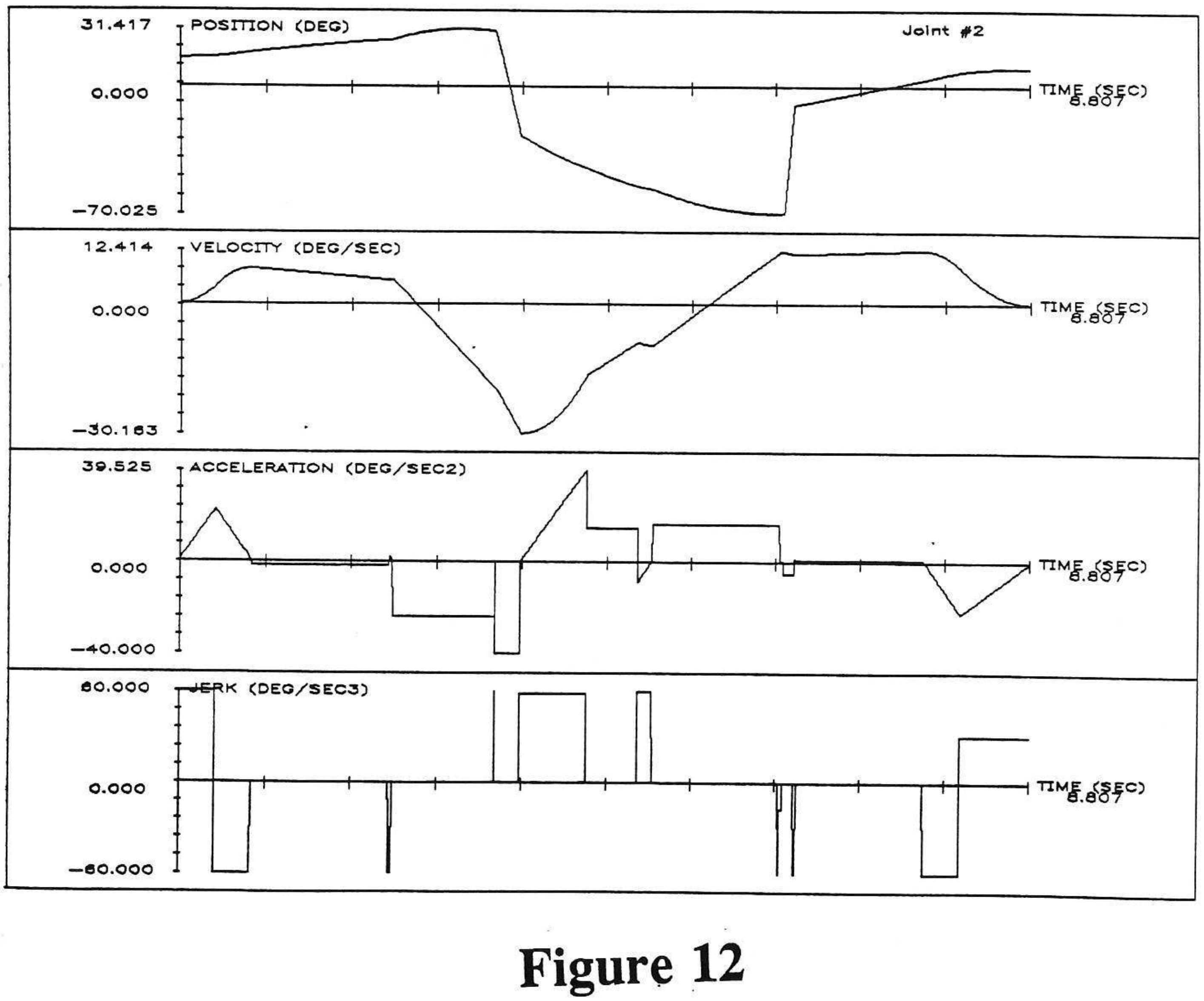

31.417
POSITION (DEG)
Joint #2
0.000
TIME (SEC)
8.807
-70.025
12.414
VELOCITY (DEG/SEC)
0.000
TIME (SEC)
8.807
-30.163
39.525
ACCELERATION (DEG/SEC2)
0.000
TIME (SEC)
8.807
-40.000
60.000
JERK (DEG/SEC3)
0.000
TIME (SEC)
8.807
-60.000

Figure 12

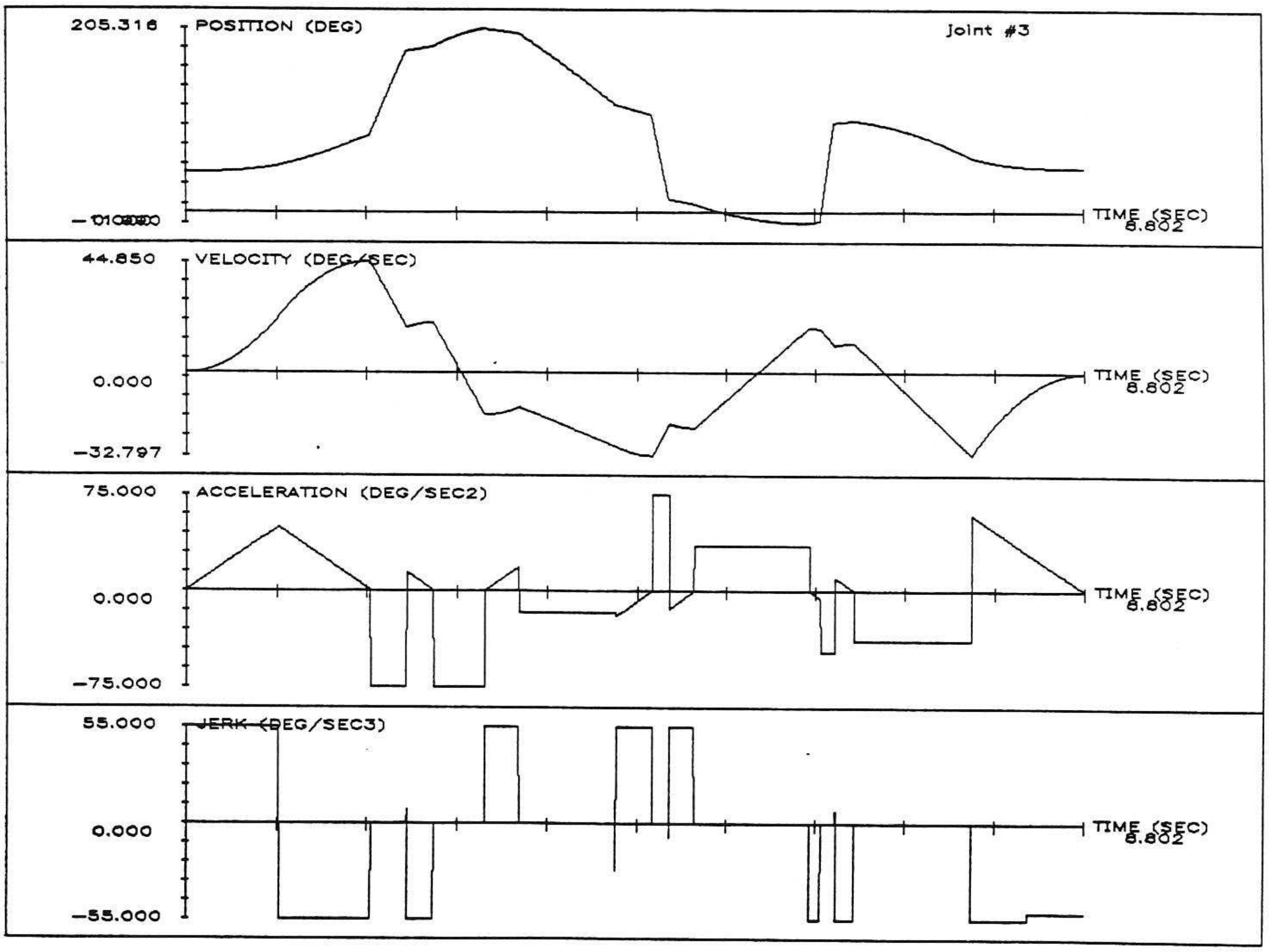
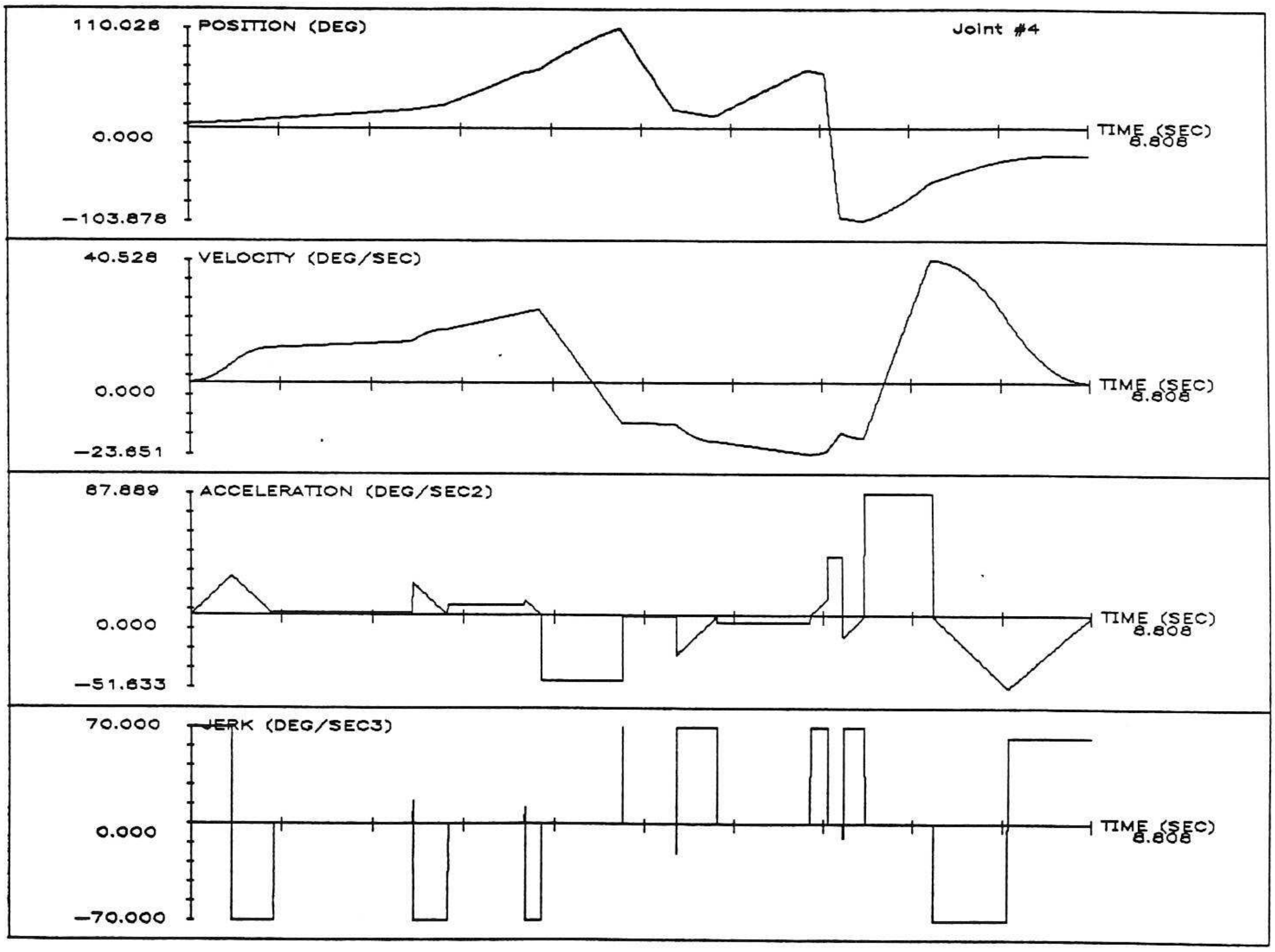
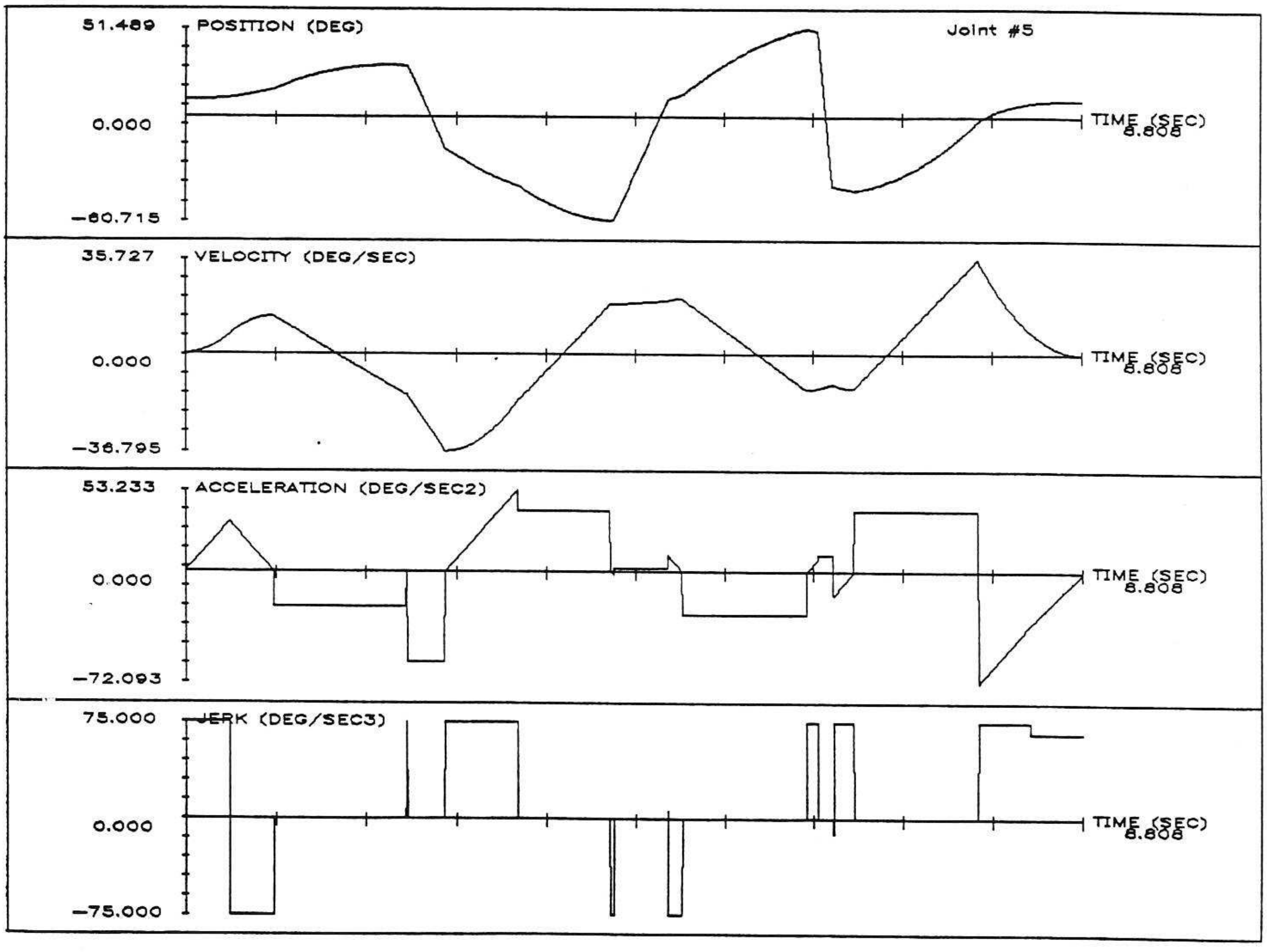

205.316
POSITION (DEG)
Joint #3
-0.000
TIME (SEC)
8.802
44.850
VELOCITY (DEG/SEC)
0.000
TIME (SEC)
8.802
-32.797
75.000
ACCELERATION (DEG/SEC2)
0.000
TIME (SEC)
8.802
-75.000
55.000
JERK (DEG/SEC3)
0.000
TIME (SEC)
8.802
-55.000

Figure 13



110.026
POSITION (DEG)
Joint #4
0.000
TIME (SEC) 8.808
−103.878
40.526
VELOCITY (DEG/SEC)
0.000
TIME (SEC) 8.808
−23.651
87.889
ACCELERATION (DEG/SEC2)
0.000
TIME (SEC) 8.808
−51.633
70.000
JERK (DEG/SEC3)
0.000
TIME (SEC) 8.808
−70.000

Figure 14



51.489
POSITION (DEG)
Joint #5
0.000
TIME (SEC) 8.808
-60.715
35.727
VELOCITY (DEG/SEC)
0.000
TIME (SEC) 8.808
-36.795
53.233
ACCELERATION (DEG/SEC2)
0.000
TIME (SEC) 8.808
-72.093
75.000
JERK (DEG/SEC3)
0.000
TIME (SEC) 8.808
-75.000

Figure 15