Monograph:
Zalzala, Ali. M.S. and Morris, A.S. (1989) Real-Time Robot Motion: The VLSI Implementation Via Transputers. Research Report. Acse Report 364 . Dept of Automatic Control and System Engineering. University of Sheffield

Real-Time Robot Motion: The VLSI Implementation Via Transputers

by:

Ali M. S. Zalzala and Alan S. Morris

Department of Control Engineering,
University of Sheffield,
Mappin Street, Sheffield S1 3JD,
United Kingdom

Research Report # 364

June 1989

Abstract

An on-line trajectory generator has been implemented for the control of robot manipulators. The computational complexities associated with such a task have been reduced significantly through distributing it on a multiprocessor system, where the minimum-time motion of the robot is planned in real-time. In particular, the distributed structure employing the transputer machines emphasise the practicality and efficiency of the proposed system. Simulation results of a case study are presented for a PUMA 560 robot manipulator.

1. Introduction

The recent literature in robotics and automation research shows a great interest in the problem of *Automated Trajectory Planning,* which is applicable to a wide range of applications including robot manipulators and mobile vehicles [3]. The great challange encountered is in the planning of such a trajectory for real time applications [7]. This task has recently been made possible by the rapid availability of very fast and inexpensive computers, forming the integrated environment needed to construct the planner.

One important characteristic of the new generation of robots is the presence of intelligent capabilities. This has recently being supported by the rapid development of both computer systems and sensory equipment. The latter could be considered as the information interface of the robot with the outside world, while the former stands for its working brain. The aspect of artificial intelligence should be high on the scientific research priorities if the robot is to be seen as a standalone machine. The availability of such an intelligent machine for the market would have a great impact on the automated assembly-line technology [9], which would have the capabilities of performing small-scale batch jobs and equipment repair in addition to large-scale applications. Due to recent developments in robotic applications, precise and high speed motion is required to accomplish a specific task. The Trajectory Control problem (TC) of robot manipulators is concerned with the movement of an object from one point in space to another. However, once the concept of Intelligent Robots is introduced, the TC problem must be addressed on-line, since the robot path is to be selected by means of intelligent sensory equipment. Nevertheless, such a procedure cannot be easily undertaken due to the inherent dynamical complexities associated with its implementation. Thus, in addition to the trajectory planning related complexities [4], the presenc of the highly nonlinear dynamic equations of motion creates a very computationally expensive problem [17]. Although several algorithmic simplifications had been presented [18], the robot dynamic equations of motion are still a vast complicated task when combined

with the requirements of minimum-time motion [14].

Two previous attempts have been made for the on-line planning of robot trajectories [5, 6]. However, in both contributions a single feasible trajectory was constructed while ignoring certain physical limitations of the manipulator. In addition, both were short of a practical implementation.

In this work, the principles of *concurrent processing* are being considered as a promising solution [11, 2, 12]. A minimum-time trajectory generator is to be distributed on a multiprocessor system, thus creating the possibility of on-line implementation. The relatively recent availability of fast and reliable general purpose processing elements such as the INMOS transputer, has enabled the practical construction of the proposed system. The presentation of this real-time system is the main contribution offered by this work.

2. Problem Statement

For the robot end-effector to track a minimum-time trajectory using the information provided by its on-board sensory equipment, the proposed motion 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 sensor's advice. The border point between these two present and next segments of the trajectory is defined as the *look–ahead point*, which defines the position and orientation of the robot hand at that specific moment. Each of these points would be selected by the sensors as the best suitable whenever needed, yielding large flexibility in the planning procedure. The planning process will consider all realistic constraints which may limit the manipulator performance, namely angular positions, velocities, accelerations, jerks and the actuators' torques (or forces).

3. The Distributed Trajectory Generator (DTG)

3.1. General Formulation

Planning of robot motion in made in the configuration space, where the joint-trajectories would be composed of successive polynomial segments between every two look-ahead points. The time-minimization problem addressed depends on fitting a combined spline of both cubic and quadratic polynomials for each segment of the trajectory, and further varying the trajectory produced in an attempt to optimise the travelling time. Initial continuity conditions are guaranteed via cubic splines. However, once a time-optimal segment has been detected, it is linked with the previously

planned segment by an approximated quadratic polynomial. Detailed formulation of the proposed method had been reported elsewhere [20, 21, 22, 19], while the main intention given here is to its practical implementation on an actual multiprocessor system. For the planning procedure to be sufficiently accurate, planning of a segment i must be completed before the minipulator completes traversing the previous segment, i–1. Hence, the condition

$$t^{i}_{exec} \leq t^{i-1}_{travel} \tag{1}$$

should be maintained to guarantee continuity of motion, where,

$t^{i-1}_{travel} \equiv$ *the travelling time between points* i–1 and i , *and,*

$t^{i}_{exec} \equiv$ *planning time of a movement between points* i and i+1 ,

3.2. The Distributed Algorithm

The minimum-time planning method is composed of the following processes:

*P*1 : Transformation of a single point from the cartesian space to the configuration space of the robot, via the *inverse kinematics algorithm* [16].

*P*2 : Spline-fit and optimisation procedures, performed on alternative segments of motion, until an optimal one is detected.

*P*3 : Quadratic approximation spline, joining the present optimal segment with the previous one.

*P*4 : Checking for the actuators' input violation via the *inverse dynamics algorithm* [10].

The process *P*1 is disregarded in the implementation, since it has a minor impact on the total execution time as compared to other processes involved, specially when the transformation is computed only once for each look-ahead segment.

3.2.1. Levels of Concurrency

The proposed distributed algorithm is illustrated in figure (1), for which concurrency in the formulation is exploited at two distinct levels:

A. Global level: treating each joint of the manipulator independently, by applying processes *P*2 and *P*3 simultaneously for each.

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

To accommodate for the above levels of concurrency, three types of processing units are constructed, as follows:

The Supervisor Unit (SPU): which controls the whole network of processors for all N joints.

The Intelligent Control Unit (ICU): which controls the optimisation process of each joint.

The Optimisation Unit (OPU): which constructs a single minimum-time segment.

The Quadratic Approximation Unit (QAU): which preserves the continuity of motion between all successive segments of each joint.

Employing the processing units described above, the *DTG* system can be constructed as shown in figure (2), where N denotes the number of degrees of freedom of a given manipulator, while M represents the number of optimisation modules available for each joint.

While the global level of concurrency is readily presented by the illustration of figure (2), the local level of concurrency is controlled exclusively by each *ICU*. Hence, the M optimisation units available for each joint are activated in whole or in part, once or more, depending on the execution time allowed for planning the segment i, (i.e. the value t^{i-1}_{travel} of *eqn.*(1)). Thus, the total execution time allowed would be

$$t^{i}_{exe} = (p.r)\ \nabla_{OPU} + \nabla_{QAU} \tag{2}$$

where,

$$r \equiv \textit{number of concurrent OPUs (M or less)},$$

$$p \equiv \textit{number of successive r-member optimisations},$$

$$\nabla_{OPU} \equiv \textit{execution time of a single OPU}, \text{ and}$$

$$\nabla_{QAU} \equiv \textit{execution time for a QAU}.$$

Hence, the total number of optimisation phases to be conducted by the *ICU* is

$$N_{phase} = p.r = \frac{t^{i}_{exe} - \nabla_{QAU}}{\nabla_{OPU}} \tag{3}$$

Thus, the function of the *ICU* is to determine the maximum allowable executions of each of the *OPUs*, which leads in turn to the best possible minimisation of time.

Although the available planning time would allow for a N_{phase} number of optimisation processes to be performed, a tolerance for the minimum time sought is to be setup by the user, thus terminating the *ICU* action once the required optimality is

obtained. This would prevent any unnecessary executions to occur, leading to a better reservation of the available utilities.

One individual case of great importance is that of having

$$(t^{i}_{exe} - \nabla_{QAU}) < \nabla_{OPU} \tag{4}$$

yielding $N_{phase}<1$, which would prevent the simplest task of construction of one single segment. In such a situation, the *ICU* is instructed to adjust the value of t^{i}_{exe}, (and accordingly that of t^{i-1}_{travel}) to accommodate for the case of $N_{phase}=1$.

The function of each of the processing units is illustrated by figures (3) through (6).

4. Practical VLSI Implementation

A practical multiprocessor system has been constructed utilizing the *INMOS T800 transputer* [13] as its basic processing element. The source code for each of the processing units designed in the previous section has been written in the *parallel C* programming language [1].

4.1. Mapping the DTG on the Transputer Network

The real-time trajectory tracker is to be implemented for the *Unimation PUMA 560* robot manipulator, with six degrees-of-freedom (i.e. N=6). Hence, considering the distributed formulation of figure (2), a number of 6 processors would be reserved for each joint of the robot to perform the optimisation (i.e. M=6). This would put the total number of processors required at a total of 45 transputers. It should be noted, however, that the *QAU* function follows the termination of that of the *ICU*. Therefore, only one transputer would be adequate to accommodate for both units for each joint. Nevertheless, due to the limited number of transputers available for the Parallel Processing Laboratory at the department, the *DTG* system has been mapped for a single joint, as shown in figure (7). Planning of all other 5 joint-trajectories would be performed successively through the *SPU*.

In order to enable the *ICUs* to decide on the computational complexity of planning each joint-segment (i.e. N_{phase} of eqn(4)), the execution times of both the *OPU* and *QAU* are found to be ∇_{OPU}=9.1 msec and ∇_{QAU}=2.7 msec, respectively.

4.2. The Communications Prospective

The parallel C language employed as the programming environment does not support the (PAR) structure, as used in the *OCCAM* definition for the concurrent execution of several modules [15]. Such a structure is of great importance when a single processor is to communicate with several others simultaneously, as is the case between each *ICU* and its corresponding *OPUs*.

Hence, while sending two $3x1$ vectors A and B of floating-point numbers concurrently would be represented in *OCCAM* as follows:

```
PAR
    Channel_1 ! A
    Channel_2 ! B
```

its equivalent *parallel C* structure would be

```
for (i=0; i<3; i++)  chan_out_message (4, A[i], Channel_1);
for (i=0; i<3; i++)  chan_out_message (4, B[i], Channel_2);
```

which, in addition to being executed sequentially, involves issuing and receiving more request and acknowledge bits to and from the concerned processor, respectively. This would obviously lead to a greater communications overhead.

However, one compromise is offered by the use of *semaphores* to create an execution thread for each of the communicating channels [1]. This method would provide faster acknowledgements for all communications ports in a first-in first-served manner. The following code illustrates the proposed concept.

```
main (.....)
.....;
{
     InP = in_ports;
     OutP = out_ports;
     int   i;
     extern void GetData();

     sema_init (&buf_free, 1);
     chans = Number_Of_Inputs;

     for (i=0; i<=chans; i++)
         thread_create (GetData, 50*sizeof(int), 1, i);
}

void GetData(i);
int   i;
{
     int   bytes;

     for (;;)
         {
           chan_in_word (&bytes, InP[i]);
           sema_wait (&buf_free);
           chan_in_message (bytes, &buf[0], InP[i]);
           CopyData(bytes, buf[0]);
           sema_signal (&buf_free);
         }
}

void CopyData(bytes, buf)
int   bytes;
char  buf[1024];
{
     /* Copy received data to its destination */
}
```

It should be mentioned that the communication between each *ICU* and three of its *OPUs* is made through the other *OPUs* in the network, due to the 4-link limitation imposed by the transputer hardware design. This is unavoidable although such a structure doubles the communications burden. The same principle applies for the interconnections between each of the *ICUs* and the *SPU*.

4.3. Simulation of a Case Study

Choosing a 8-point trajectory for the *PUMA* manipulator as the proposed sensory-detected look-ahead points, simulation results can be obtained. The outcome of planning three look-ahead segments are included in the following tables, while the minimum-time trajectories constructed for the first three joints of the *PUMA* are shown in figures (8), (9) and (10).

Table (1) : Results of Planning the 2nd Segment

Joint #	ICU Instructions		Optimality Requirements	
	N_{phase}	Planning time (sec)	N_{phase}	Planning time (sec)
1	80	0.7343	3	0.0273
2	80	0.7343	3	0.0273
3	80	0.7343	3	0.0273
4	80	0.7343	3	0.0273
5	80	0.7343	5	0.0455
6	80	0.7343	3	0.0273
Planning Time = 0.737 seconds Motion Time = 1.422 seconds				

Table (2) : Results of Planning the 4th Segment

Joint #	ICU Instructions		Optimality Requirements	
	N_{phase}	Planning time (sec)	N_{phase}	Planning time (sec)
1	119	1.085	4	0.036
2	119	1.085	4	0.036
3	119	1.085	4	0.036
4	119	1.085	6	0.055
5	119	1.085	5	0.046
6	119	1.085	5	0.046
Planning Time = 1.088 seconds Motion Time = 0.948 seconds				

Table (3) : Results of Planning the 8th Segment

Joint #	ICU Instructions		Optimality Requirements	
	N_{phase}	Planning time (sec)	N_{phase}	Planning time (sec)
1	19	0.174	5	0.055
2	19	0.174	4	0.036
3	19	0.174	5	0.055
4	19	0.174	3	0.027
5	19	0.174	5	0.055
6	19	0.174	4	0.036
Planning Time = 0.177 seconds Motion Time = 2.423 seconds				

4.4. The Dynamic Considerations

Although the coupling effects of the robot joints were not considered in the design presented (i.e. process *P*4 of section 3.2), it could be easily accommodated for once an efficient distributed algorithm for the inverse dynamics is formulated. One practical implementation had been reported [23, 24], where the equations of motion could be solved in 2.46 *milliseconds* for a 4-transputer configuration, or alternatively at

the much faster rate of 26 *microseconds* for a configuration involving a network of 26 transputers. Hence, assuming a 28 *millisecond* control cycle for the *PUMA* [8], planning the 4th look-ahead segment (Table (2)) would require 0.083 seconds and 0.880 milliseconds to accommodate for the dynamic effects considering the 4-transputer and the 26-transputer configurations, respectively. Thus, the planning time reported would increase from 0.055 seconds to 0.138 seconds and 0.056 seconds, respectively, which is well below the imposed execution time of 1.088 seconds. Similar arguments could be made for all other segments.

5. Conclusion

A fast minimum-time trajectory generator for robot manipulators has been presented, deploying a highly parallel multiprocessor system. The practicality of such an on-line scheme for robot control has been proved through its implementation on an actual multiprocessing system utilizing the INMOS T800 transputer. It is the authors' belief that the recent advances in VLSI technology would present the ultimate solution for the computationally expensive problem of robot minimum-time trajectory planning and control in real-time applications. This work is seen as one contribution towards accomplishing that aim.

References

1]. 3L,(1988), *Parallel C User Guide*, 3L Ltd, Scotland, United Kingdom.

[2]. BERTSEKAS, D. P. AND TSITSIKLIS, J. N., (1989). *Parallel and Distributed Computation : Numerical Methods*, Prentice-Hall.

[3]. BOLLES, R. C. AND ROTH, B. (EDS.), (1988). *Robotics Research, The 4th Int. Symp.*, MIT Press.

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

[5]. 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.

[6]. 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.

[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]. HWANG, K. AND BRIGGS, F. A., (1985). *Computer Architecture and Parallel Processing*, McGraw-Hill.

[12]. INMOS,, (1988). *Communicating Process Architecture*, Prentice-Hall.

13]. INMOS,(1988), *IMS T800 Transputer*, Technical Note #6.

[14]. 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.

15]. MAY, D. AND SHEPHERD, R., (1988). "The Transputer Implementation of Occam," INMOS Technical Note #21.

[16]. PAUL, R. P., SHIMANO, B., AND MAYER, G. E., (1981). "Kinematic Control Equations for Simple Manipulators," *IEEE Trans. Syst., Man, Syber.*, vol. SMC-11, pp. 449-55.

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

[18]. VUKOBRATOVIC, M. AND STOKIC, D., (1983). "Is Dynamic Control Needed in Robotic Systems, and, if So, to What Extent ?," *Int. J. Robotics Research*, vol. 2, no. 2, pp. 18-34.

19]. 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, Department of Control Engineering, University of Sheffield, United Kingdom.

[20]. ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "An On-Line Minimum-Time Trajectory Generator for Intelligent Robot Manipulators," *To appear, IMC Sixth Conference on Advanced Manufacturing Technology*, 31st August - 1st September, Dublin, Republic of Ireland.

21]. ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "An On-Line Distributed Minimum-Time Trajectory Generator for Intelligent Robot Manipulators," Research Report #358, Department of Control Engineering, University of Sheffield, United Kingdom.

[22]. ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "Minimum-Time Trajectory Planning for Articulated Manipulators in Configuration-Space and Under Physical Constraints," *To appear, IMA Conf. on Robotics: Applied Mathematics and Computational Aspects, 12-14 July*, Loughborough University of Technology, United Kingdom.

23]. 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.

[24]. ZALZALA, A. M. S. AND MORRIS, A. S., (1989). "A Distributed Pipelined Architecture of Robot Dynamics with VLSI Implementation," *Submitted for publication, Int. J. Robotics and Automation.*

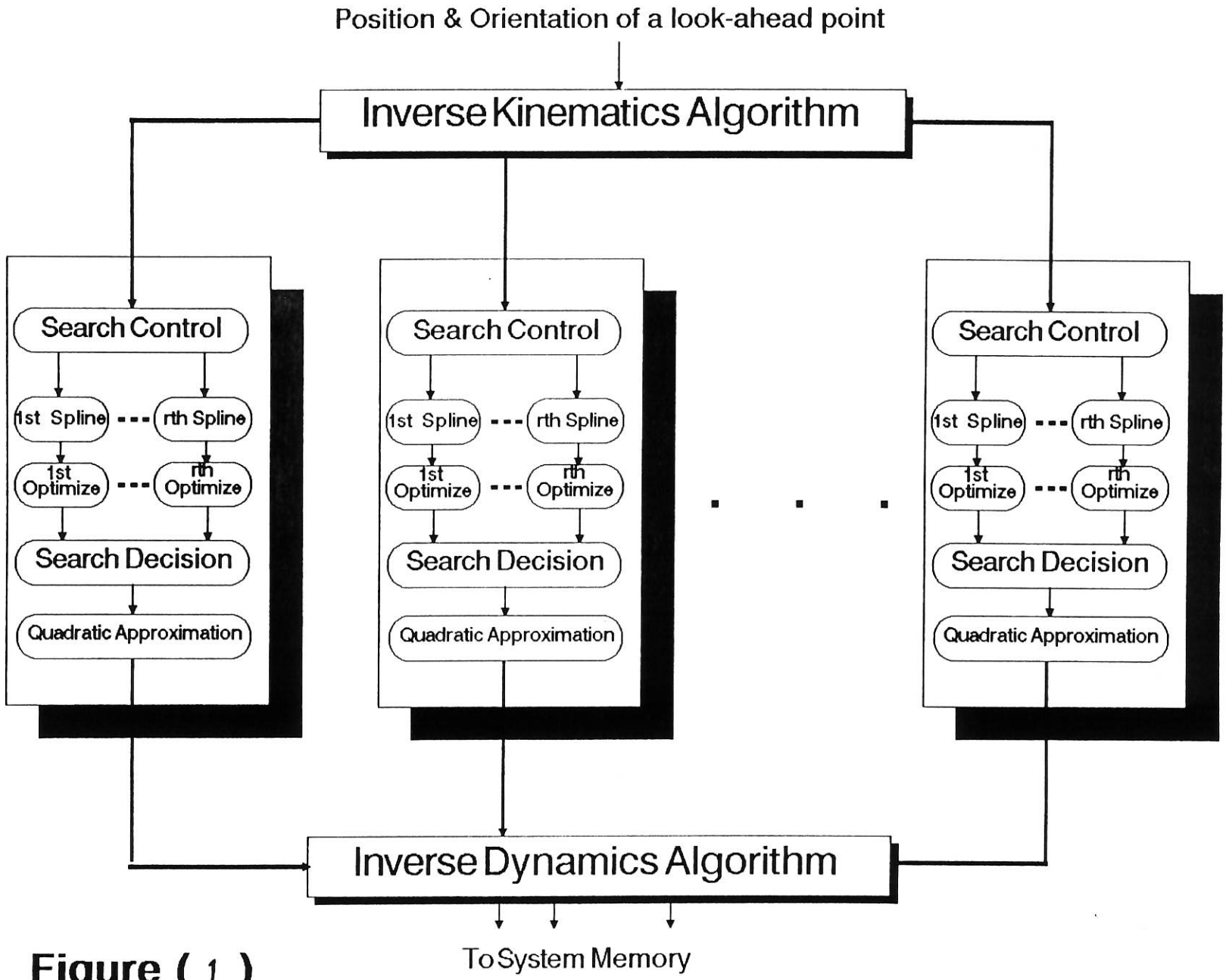
Position & Orientation of a look-ahead point
Inverse Kinematics Algorithm
Search Control
1st Spline
rth Spline
1st Optimize
rth Optimize
Search Decision
Quadratic Approximation
Search Control
1st Spline
rth Spline
1st Optimize
rth Optimize
Search Decision
Quadratic Approximation
Search Control
1st Spline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

Figure (1)

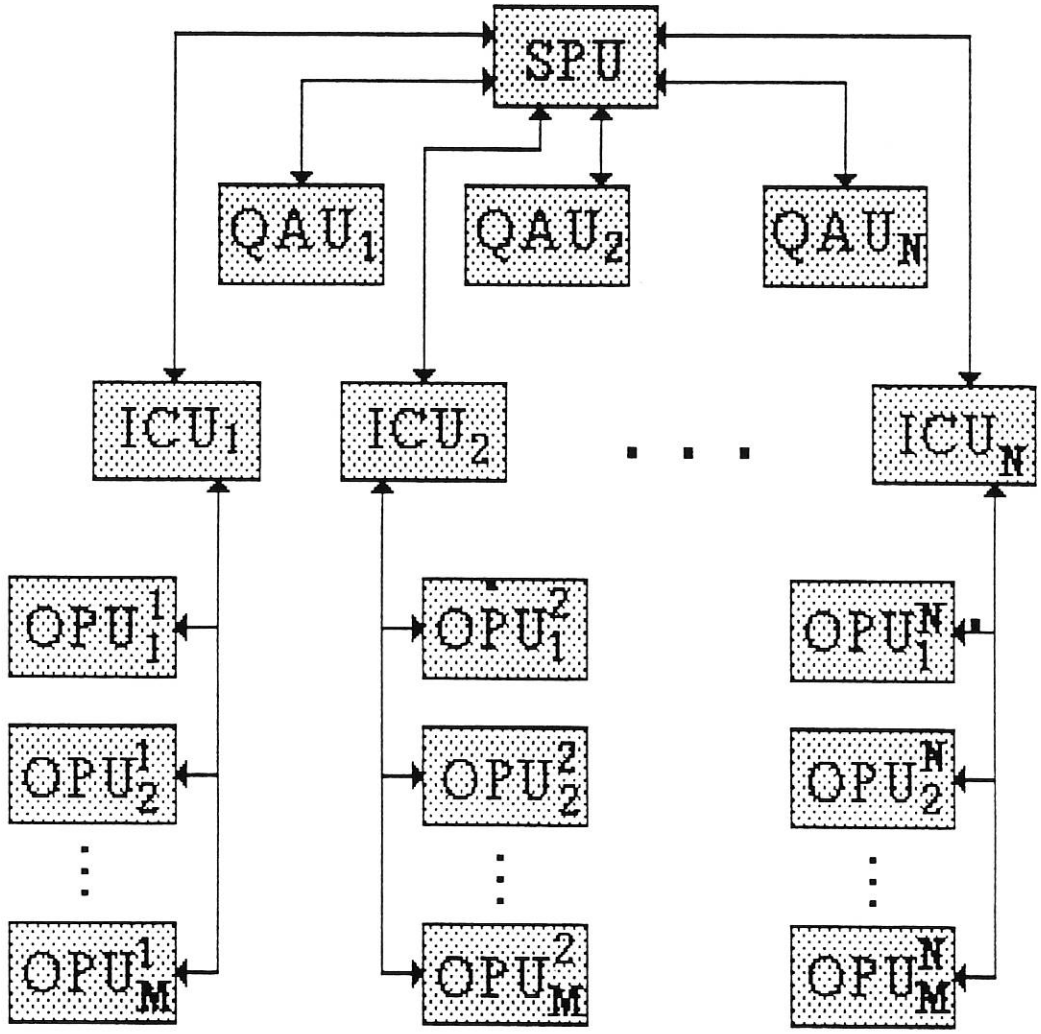

Figure (2) : The DTG Algorithm

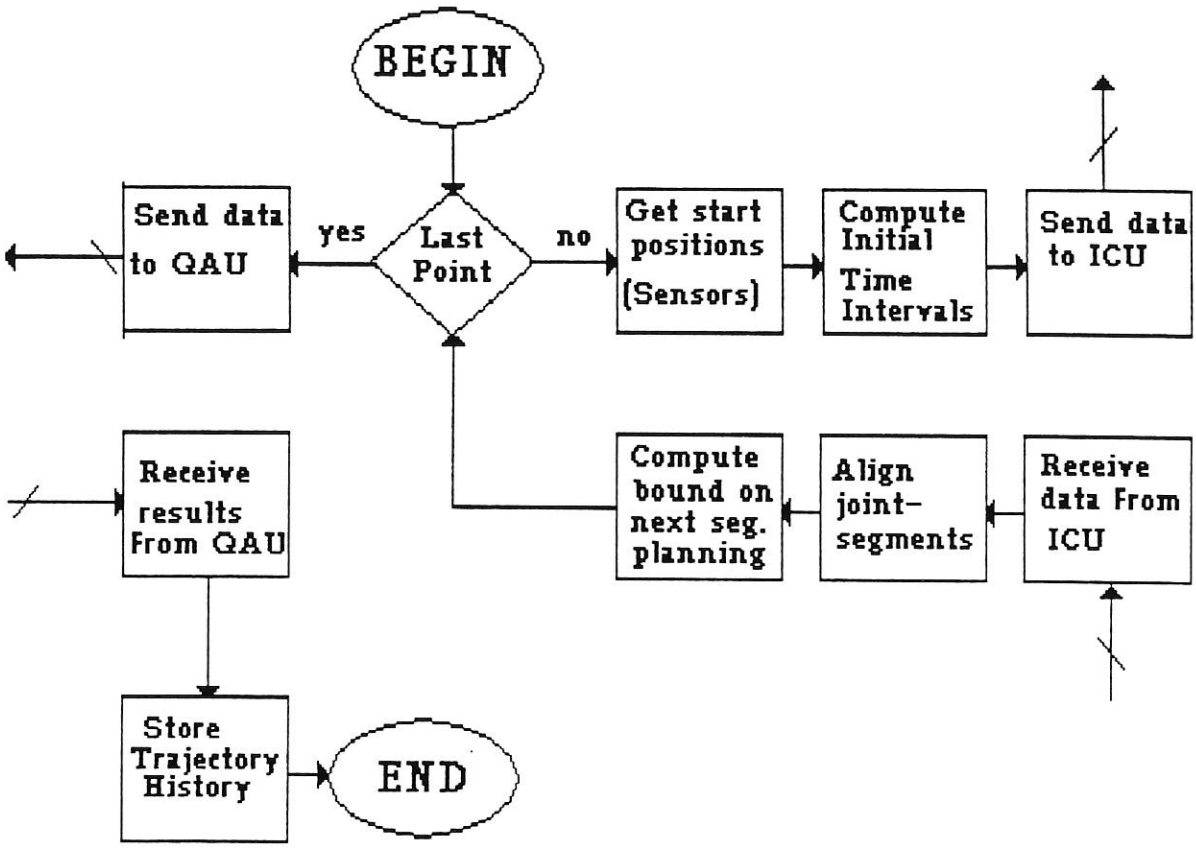

Figure (3) : The Supervisor Unit

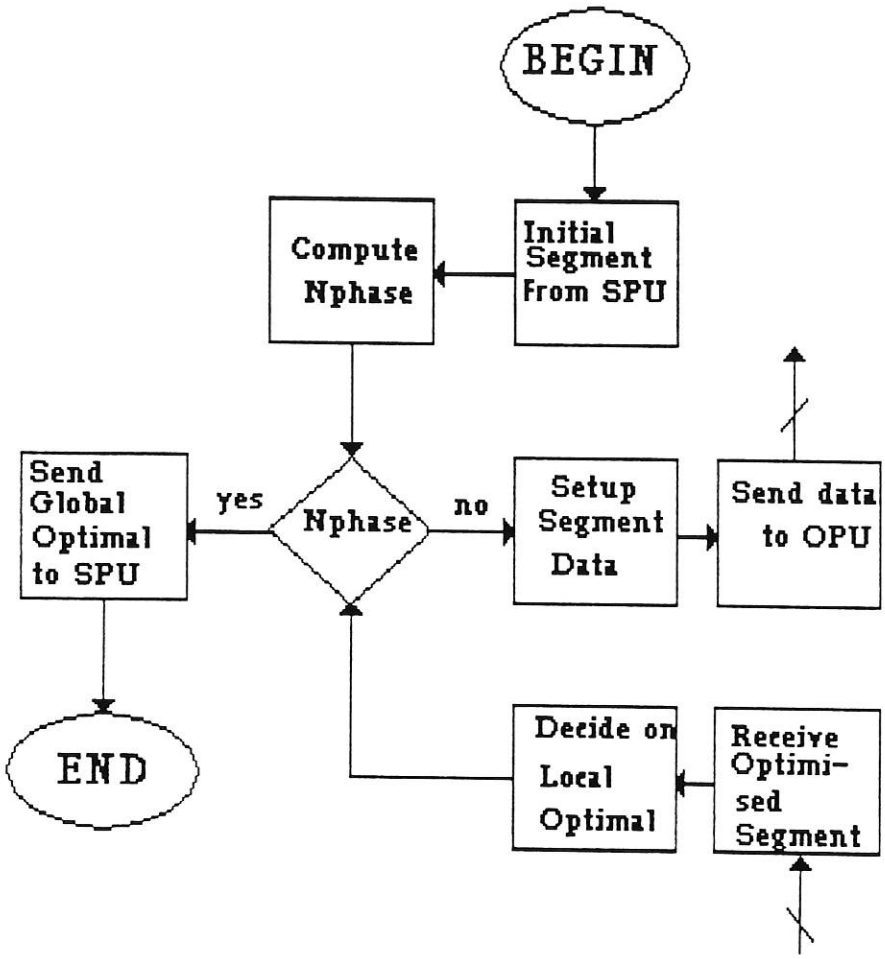

Figure (4) : The Intelligent Control Unit

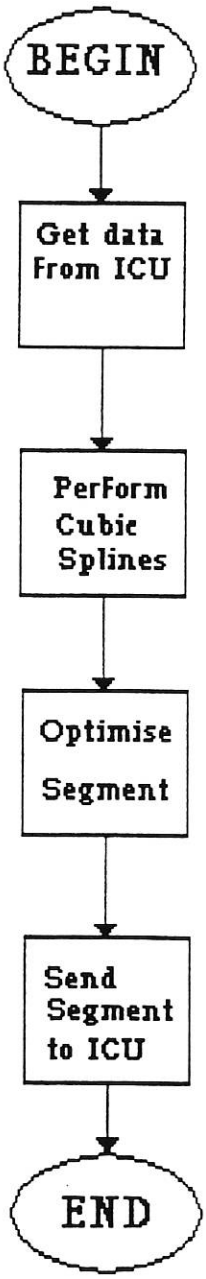

Figure (5): The Optimisation Unit

Figure (6) : The Quadratic Approximation Unit

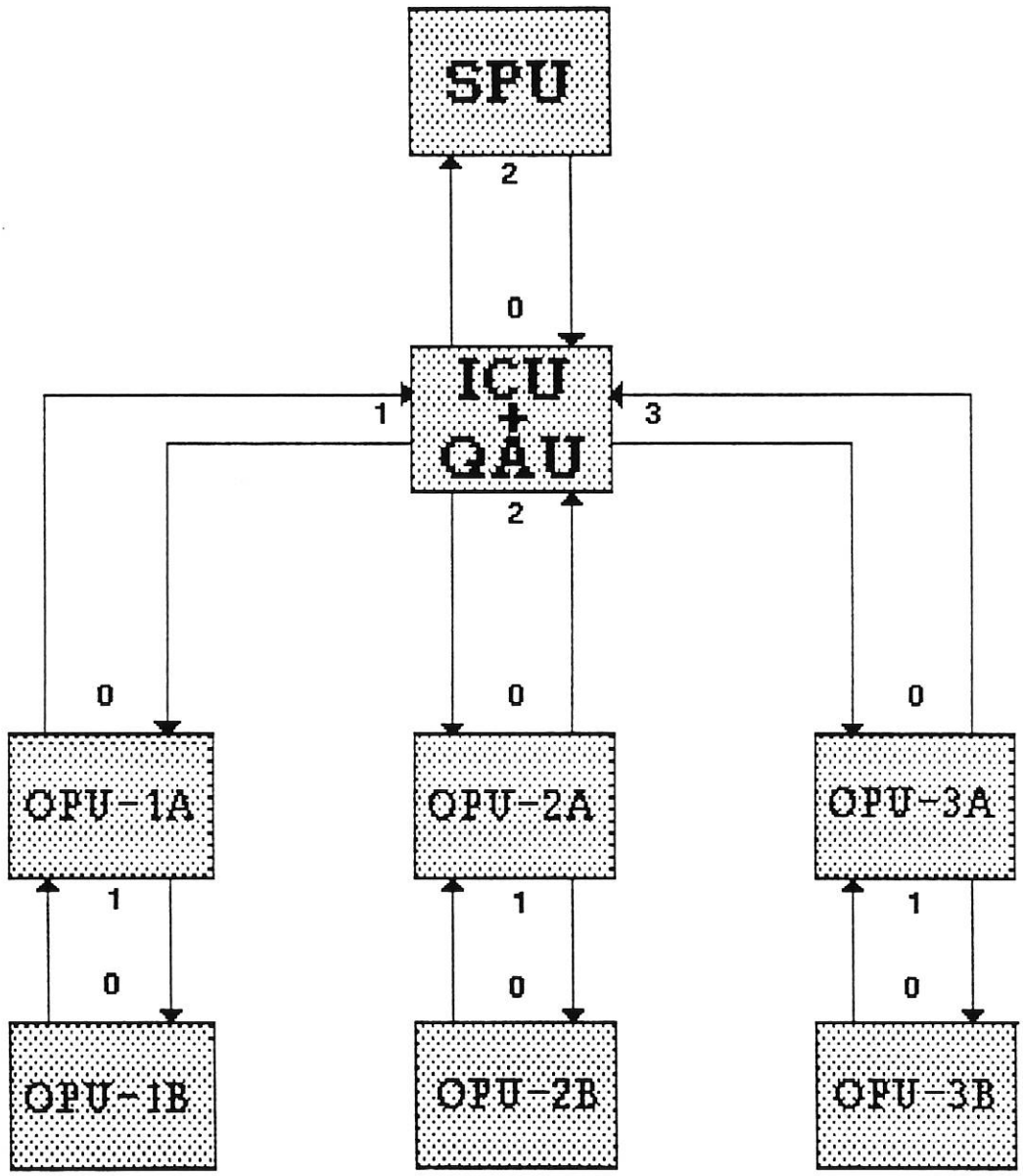

Figure (7) : Transputer Network Configuration

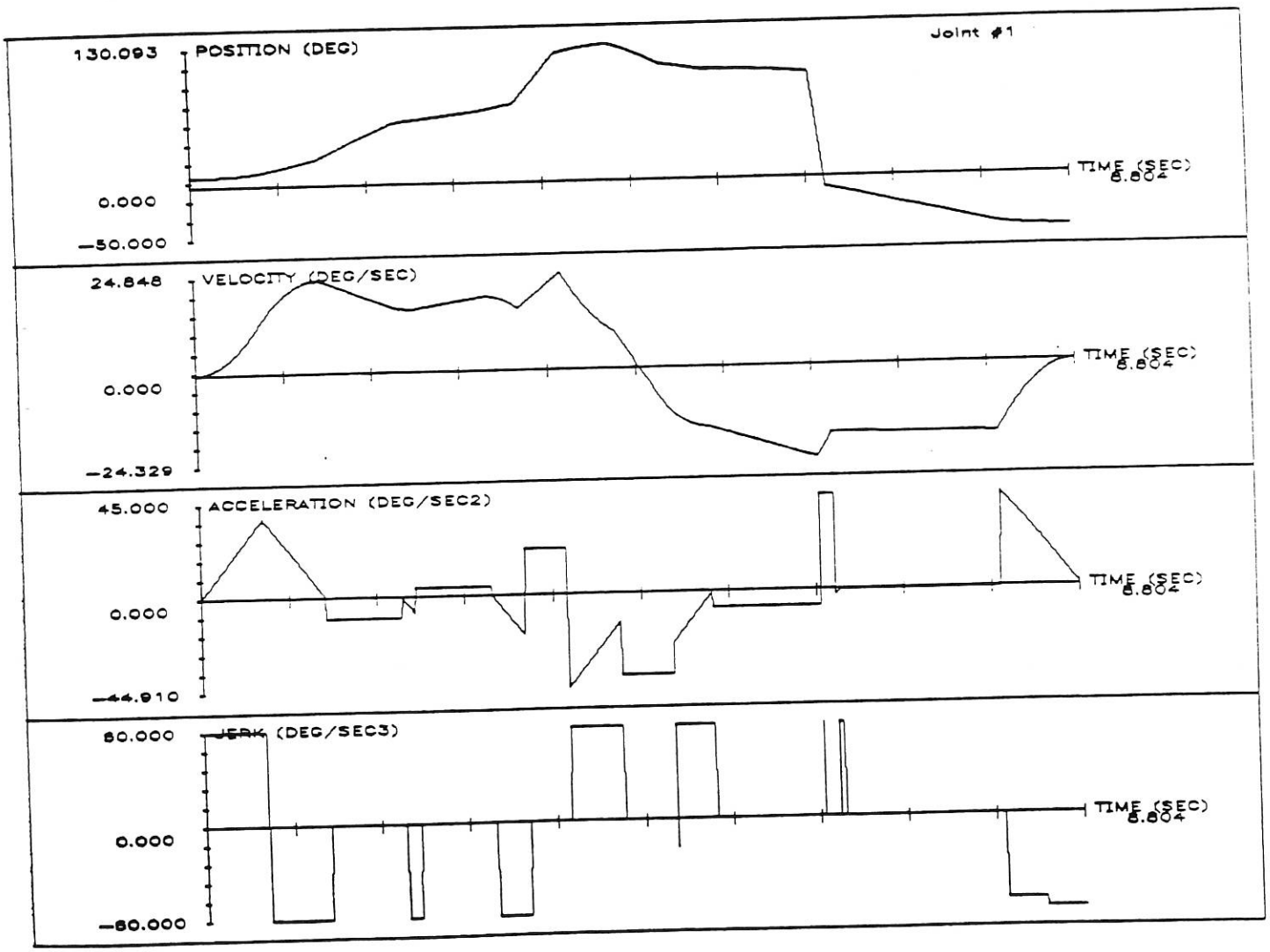

Figure 8 : Minimum-Time Trajectory for Joint #1

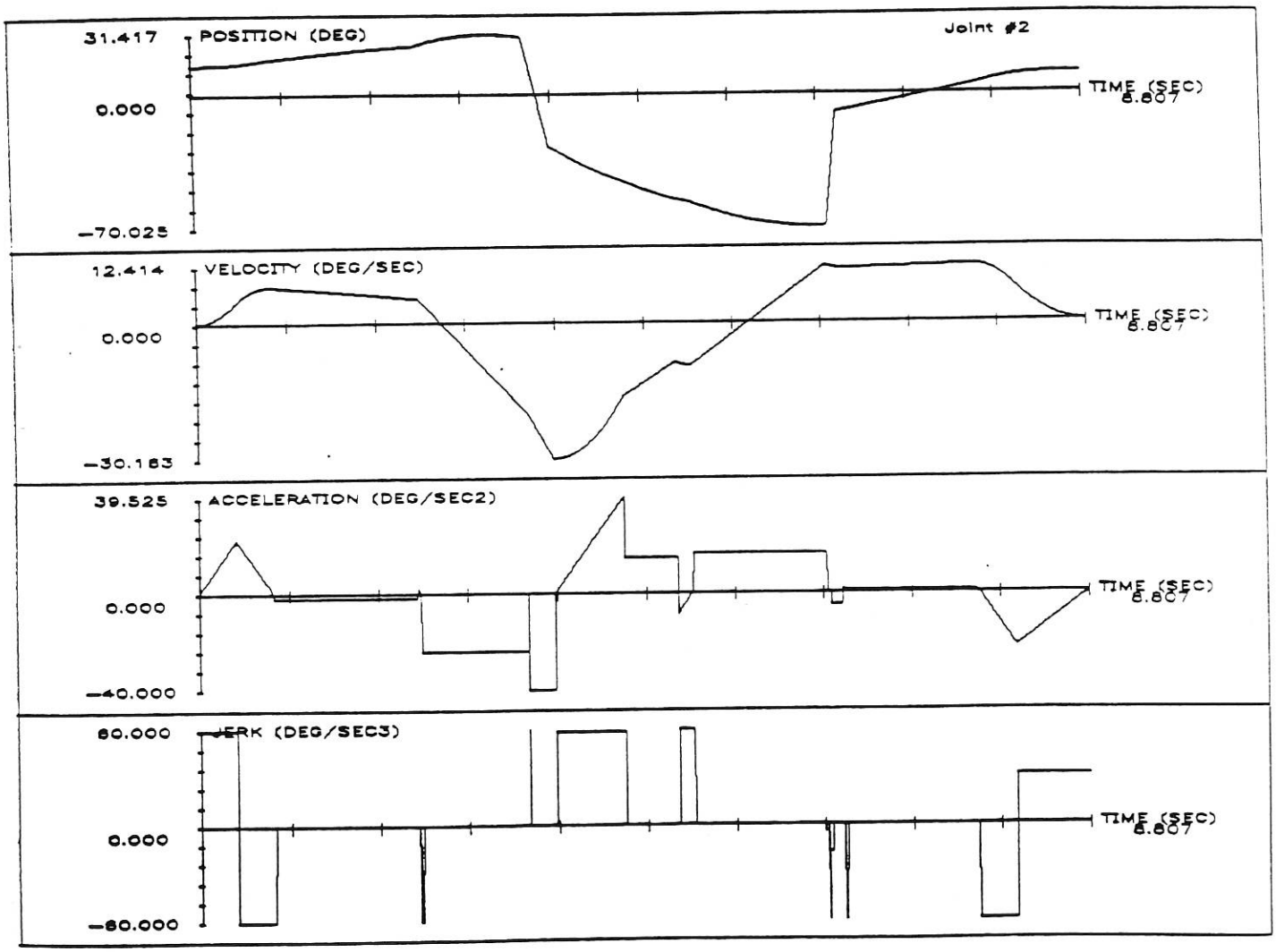

Figure 9 : Minimum-Time Trajectory for Joint #2

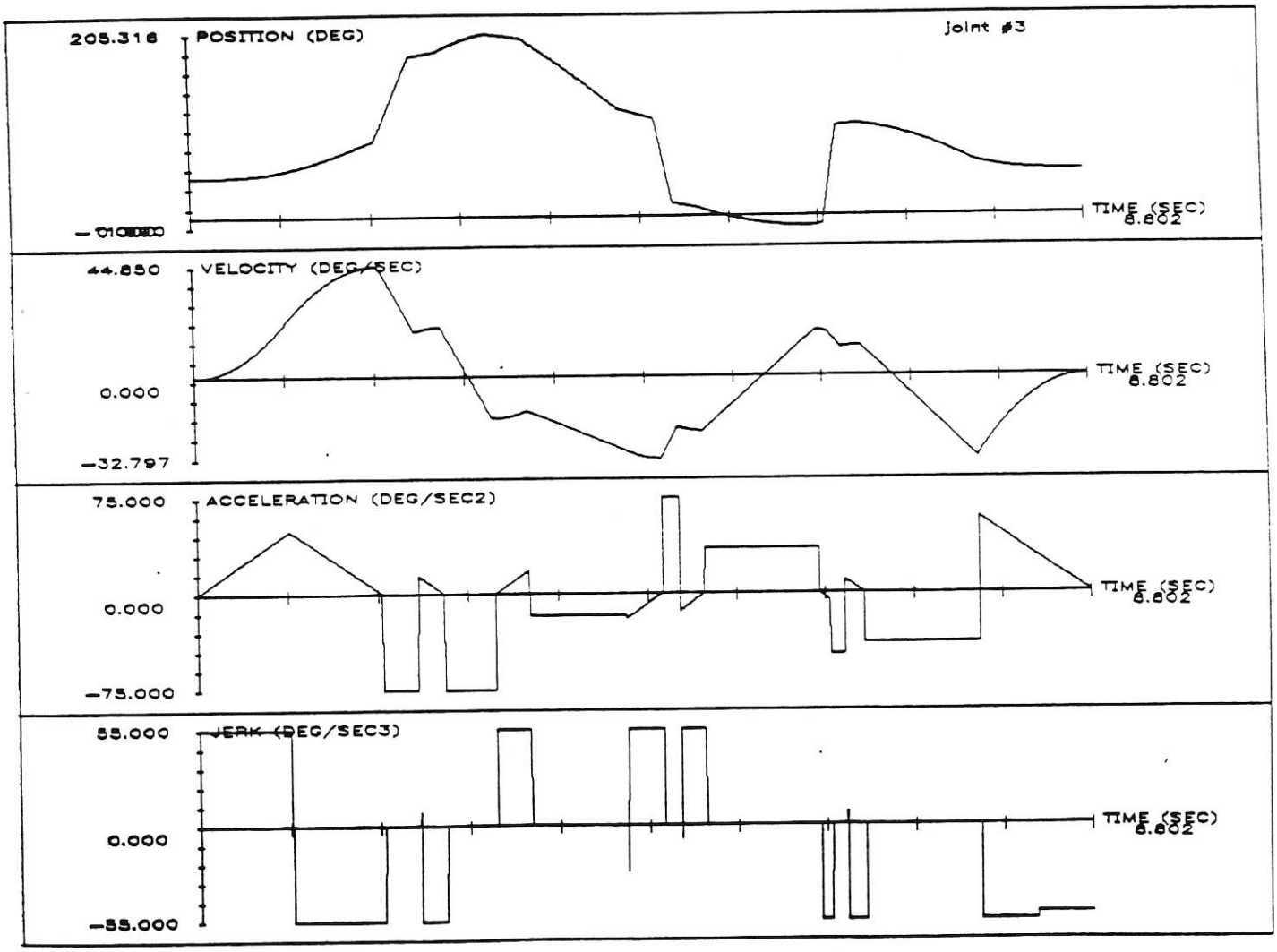

Figure 10 : Minimum-Time Trajectory for Joint #3