

**MINIMUM-COST CONTROL OF ROBOTIC MANIPULATORS  
WITH GEOMETRIC PATH CONSTRAINTS**

by

Neil David McKay

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Electrical Engineering)  
in The University of Michigan  
1985

Doctoral Committee:

Associate Professor Kang G. Shin, Chairman  
Professor N. Harris McClamroch  
Associate Professor Trevor N. Mudge  
Professor Richard A. Volz



## **ACKNOWLEDGEMENTS**

I would like to thank my advisor, Kang Shin, for his guidance and occasional prodding over the past five years. I would also like to thank my other committee members for their advice, my parents for their moral (and occasionally financial) support, and the other graduate students who live in the 1500 wing of East Engineering for all the good times they provided.

The work described in this thesis was supported in part by NSF Grant No. ECS-8409938 and the US AFOSR Contract No. F49620-82-C-0089. Any opinions, findings, and conclusions or recommendations are those of the author and do not necessarily reflect the view of the funding agencies.





## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b> .....	ii
<b>LIST OF FIGURES</b> .....	vi
<b>LIST OF TABLES</b> .....	ix
<b>CHAPTER</b>	
<b>1. INTRODUCTION AND LITERATURE SURVEY</b> .....	1
<b>2. MATHEMATICAL PRELIMINARIES</b> .....	12
2.1. Tensors and Tensor Notation .....	12
2.2. Riemannian Spaces and Properties of Curves .....	15
2.3. Robot Arm Dynamics .....	19
2.4. Derivation of the Lagrangian Dynamic Equations .....	20
<b>3. PROBLEM STATEMENT</b> .....	27
<b>4. TRAJECTORY PLANNING</b> .....	32
4.1. Parameterization of the Robot Dynamic Equations .....	36
4.2. The Phase Plane Method .....	38
4.2.1. Derivation of Pseudo-Acceleration Limits .....	38
4.2.2. Formulation of the Optimal Control Problem .....	41
4.2.3. Phase Plane Plots .....	44
4.2.4. Determination of Optimal Trajectories .....	50
4.2.5. Numerical Examples .....	57
4.3. The Dynamic Programming Method .....	76
4.3.1. Problem Formulation .....	77
4.3.2. Trajectory Planning Using Dynamic Programming .....	78
4.3.3. Case of Non-Interacting Torque Bounds .....	79
4.3.4. Case of Interacting Torque Bounds .....	84
4.3.5. Accommodation of Jerk Constraints .....	87
4.3.6. Algorithm Complexity .....	89



4.3.7. Algorithm Speedup .....	92
4.3.8. Convergence Properties .....	94
4.3.9. Numerical Examples .....	112
4.4. Trajectory Planning by Iterative Improvement .....	154
4.4.1. The Perturbation Trajectory Improvement Algorithm .....	154
4.4.2. Computational Requirements .....	159
4.4.3. Numerical Examples .....	160
<b>5. SELECTION OF GEOMETRIC PATHS .....</b>	<b>183</b>
5.1. Near-Minimum Time Path Planning .....	184
5.1.1. A Special Case .....	185
5.1.2. Traversal Time Bounds .....	188
5.1.3. Approximate Minimum Time Paths .....	197
5.2. Numerical Examples .....	201
<b>6. COMPENSATION FOR DYNAMIC UNCERTAINTIES   .....</b>	<b>215</b>
6.1. Calculation of Dynamic Coefficient Errors .....	217
6.2. Calculation of Torque Error Bounds .....	222
6.3. Numerical Examples .....	231
<b>7. AUTOMATIC GENERATION OF TRAJECTORY   PLANNERS .....</b>	<b>257</b>
7.1. Trajectory Planning System Structure .....	258
7.2. Data Structures for Representing Geometric Paths .....	259
7.3. Selection of a Trajectory Planning Method .....	262
7.4. Generation of Dynamic Equations .....	263
7.5. Generating the Constraint Function .....	265
7.6. Ancillary Software .....	266
7.7. Work Accomplished .....	267
<b>8. CONCLUSIONS .....</b>	<b>274</b>
<b>APPENDIX .....</b>	<b>278</b>



**REFERENCES** ..... 298



## LIST OF FIGURES

Figure 4.2.1. Admissible regions of $\mu$ determined by parabolic constraints .....	65
Figure 4.2.2. Intersection of admissible regions of $\mu$ .....	66
Figure 4.2.3. Acceleration and deceleration vectors at each phase point .....	67
Figure 4.2.4. Case when accelerating and decelerating curves intersect .....	68
Figure 4.2.5. Case when accelerating and decelerating curves do not intersect .....	69
Figure 4.2.6. A complete optimal trajectory with three switching points .....	70
Figure 4.2.7. The two degree-of-freedom polar robot .....	71
Figure 4.2.8. Optimal trajectory, zero friction case .....	72
Figure 4.2.9. Admissible region, high-friction case .....	73
Figure 4.2.10. Optimal trajectory, high-friction case .....	74
Figure 4.2.11. Admissible region for cartesian manipulator .....	75
Figure 4.3.1. Curves connecting adjacent DP grid points .....	119
Figure 4.3.2. Curve connecting a grid point to an existing trajectory .....	120
Figure 4.3.3. Suboptimal trajectory with uncertainty swath .....	121
Figure 4.3.4. Minimum time phase plane plot for polar manipulator, $10 \times 10$ grid .....	122
Figure 4.3.5. Minimum time phase plane plot for polar manipulator, $40 \times 40$ grid .....	123
Figure 4.3.6a. Minimum time phase plot for PACS arm, $10 \times 10$ grid .....	124
Figure 4.3.6b. $\theta$ joint torque vs. time, $10 \times 10$ grid .....	125
Figure 4.3.6c. $r$ joint force vs. time, $10 \times 10$ grid .....	126
Figure 4.3.6d. $z$ joint force vs. time, $10 \times 10$ grid .....	127
Figure 4.3.6e. Motor voltages vs. time, $10 \times 10$ grid .....	128
Figure 4.3.7a. Minimum time phase plot for PACS arm, $40 \times 160$ grid .....	129
Figure 4.3.7b. $\theta$ joint torque vs. time, $40 \times 160$ grid .....	130
Figure 4.3.7c. $r$ joint force vs. time, $40 \times 160$ grid .....	131
Figure 4.3.7d. $z$ joint force vs. time, $40 \times 160$ grid .....	132
Figure 4.3.7e. Motor voltages vs. time, $40 \times 160$ grid .....	133
Figure 4.3.8a. Phase plot for PACS arm, minimum time-energy, $10 \times 10$ grid .....	134
Figure 4.3.8b. $\theta$ joint torque vs. time, $10 \times 10$ grid .....	135
Figure 4.3.8c. $r$ joint force vs. time, $10 \times 10$ grid .....	136
Figure 4.3.8d. $z$ joint force vs. time, $10 \times 10$ grid .....	137
Figure 4.3.8e. Motor voltages vs. time, $10 \times 10$ grid .....	138





Figure 4.3.9a. Phase plot for PACS arm, minimum time-energy, 20×40 grid .....	139
Figure 4.3.9b. $\theta$ joint torque vs. time, 20×40 grid .....	140
Figure 4.3.9c. $r$ joint force vs. time, 20×40 grid .....	141
Figure 4.3.9d. $z$ joint force vs. time, 20×40 grid .....	142
Figure 4.3.9e. Motor voltages vs. time, 20×40 grid .....	143
Figure 4.3.10a. Minimum time phase plot for PACS arm with power limit, 10×10 grid .....	144
Figure 4.3.10b. $\theta$ joint torque vs. time, 10×10 grid .....	145
Figure 4.3.10c. $r$ joint force vs. time, 10×10 grid .....	146
Figure 4.3.10d. $z$ joint force vs. time, 10×10 grid .....	147
Figure 4.3.10e. Motor voltages vs. time, 10×10 grid .....	148
Figure 4.3.11a. Minimum time phase plot for PACS arm with power limit, 20×40 grid .....	149
Figure 4.3.11b. $\theta$ joint torque vs. time, 20×40 grid .....	150
Figure 4.3.11c. $r$ joint force vs. time, 20×40 grid .....	151
Figure 4.3.11d. $z$ joint force vs. time, 20×40 grid .....	152
Figure 4.3.11e. Motor voltages vs. time, 20×40 grid .....	153
Figure 4.4.1. Results of Algorithm A. ....	163
Figure 4.4.2. Results of Algorithm A'. ....	164
Figure 4.4.3a. Phase plane plot for straight line .....	165
Figure 4.4.3b. Joint position vs. time for straight line .....	166
Figure 4.4.3c. Motor voltage vs. time for straight line .....	167
Figure 4.4.4a. Phase plane plot for geodesic .....	168
Figure 4.4.4b. Joint position vs. time for geodesic .....	169
Figure 4.4.4c. Motor voltage vs. time for geodesic .....	170
Figure 4.4.5a. Phase plane plot for joint-interpolated curve .....	171
Figure 4.4.5b. Joint position vs. time for joint-interpolated curve .....	172
Figure 4.4.5c. Motor voltage vs. time for joint-interpolated curve .....	173
Figure 4.4.6. Phase plane plot for straight line with jerk constraint, 25 points .....	174
Figure 4.4.7. Phase plane plot for straight line with jerk constraint, 50 points .....	175
Figure 4.4.8. Illustration of deadlock caused by jerk constraints .....	176
Figure 4.4.9a. Phase plane plot for straight line with jerk constraint, 50 points, velocity increment 0.1 .....	177
Figure 4.4.9b. Joint position vs. time for straight line with jerk constraint, 50 points, velocity increment 0.1 .....	178
Figure 4.4.9c. Motor voltage vs. time for straight line with jerk constraint, 50 points, velocity increment 0.1 .....	179



Figure 4.4.10a. Phase plane plot for straight line with jerk constraint, 100 points, velocity increment 0.025 .....	180
Figure 4.4.10b. Joint position vs. time for straight line with jerk constraint, 100 points, velocity increment 0.025 .....	181
Figure 4.4.10c. Motor voltage vs. time for straight line with jerk constraint, 100 points, velocity increment 0.025 .....	182
Figure 5.2.1a. Phase plane plot for straight line .....	206
Figure 5.2.1b. Joint position vs. time for straight line .....	207
Figure 5.2.1c. Motor voltage vs. time for straight line .....	208
Figure 5.2.2a. Phase plane plot for joint-interpolated path .....	209
Figure 5.2.2b. Joint position vs. time for joint-interpolated path .....	210
Figure 5.2.2c. Motor voltage vs. time for joint-interpolated path .....	211
Figure 5.2.3a. Phase plane plot for geodesic .....	212
Figure 5.2.3b. Joint position vs. time for geodesic .....	213
Figure 5.2.3c. Motor voltage vs. time for geodesic .....	214
Figure 6.3.1a. Phase plane plot for zero error. ....	239
Figure 6.3.1b. Voltage vs. time for zero error. ....	240
Figure 6.3.2a. Phase plane plot for density 12 g./cc. ....	241
Figure 6.3.2b. Joint position vs. time, 12 g./cc. ....	242
Figure 6.3.2c. Nominal and actual motor voltages, $z$ joint, 12 g./cc. ....	243
Figure 6.3.2d. Nominal and actual motor voltages, $\theta$ joint, 12 g./cc. ....	244
Figure 6.3.2e. Nominal and actual motor voltages, $r$ joint, 12 g./cc. ....	245
Figure 6.3.2f. Nominal and actual joint forces, $z$ joint, 12 g./cc. ....	246
Figure 6.3.2g. Nominal and actual joint torques, $\theta$ joint, 12 g./cc. ....	247
Figure 6.3.2h. Nominal and actual joint forces, $r$ joint, 12 g./cc. ....	248
Figure 6.3.3a. Phase plane plot for density 24 g./cc. ....	249
Figure 6.3.3b. Joint position vs. time, 24 g./cc. ....	250
Figure 6.3.3c. Nominal and actual motor voltages, $z$ joint, 24 g./cc. ....	251
Figure 6.3.3d. Nominal and actual motor voltages, $\theta$ joint, 24 g./cc. ....	252
Figure 6.3.3e. Nominal and actual motor voltages, $r$ joint, 24 g./cc. ....	253
Figure 6.3.3f. Nominal and actual joint forces, $z$ joint, 24 g./cc. ....	254
Figure 6.3.3g. Nominal and actual joint torques, $\theta$ joint, 24 g./cc. ....	255
Figure 6.3.3h. Nominal and actual joint forces, $r$ joint, 24 g./cc. ....	256
Figure 7.1.1. Overview of ATPG system .....	269
Figure 7.2.1. Robot path data structures .....	270
Figure 7.3.1. Trajectory planner structure .....	271
Figure 7.4.1. Schematic of Dynamic Equation Generator .....	272
Figure 7.6.1. ATPG structure .....	273
Figure A.1. Schematic of the first three links of the PACS robot .....	293
Figure A.2. Link coordinate frames of the PACS robot .....	294
Figure A.3. PACS actuator circuit diagram .....	295



## LIST OF TABLES

Table 4.3.1. Computation times for different grid sizes (seconds) .....	118
Table 6.3.1. Traversal times for nominal and optimal trajectories .....	236
Table 6.3.2. Minimum required voltages for nominal and actual payloads. .....	236
Table 6.3.3. Maximum required voltages for nominal and actual payloads. .....	237
Table 6.3.4. Minimum required torques/forces for nominal and actual pay- loads. ....	237
Table 6.3.5. Maximum required torques/forces for nominal and actual pay- loads. ....	238
Table A.1. Characteristics of the polar manipulator .....	291
Table A.2. Characteristics of the PACS arm .....	292



## CHAPTER 1

### INTRODUCTION AND LITERATURE SURVEY

During the past several years a great deal of attention has been focused on industrial automation techniques, especially the use of general-purpose robots. Those industries which in the past have constructed special-purpose devices for manufacturing their products are now looking at the possibility of using robots instead. Unlike special-purpose tools, robots' behavior can easily be modified, so that retooling is kept to a minimum. In some cases they can also make feasible the manufacture of a product in lots which would be too small to justify the creation of a special-purpose machine for their manufacture. Mechanical maintenance is also simpler, since there presumably would be only a few types of robots performing many different tasks. (It should be noted, though, that maintenance of special-purpose machines is replaced with maintenance of special-purpose programs.)

Since robots may be controlled in virtually any manner, one legitimately may ask how a robot should best be controlled. The obvious answer to this question is that the robot should produce as large a profit as possible per unit time. The usual assumption is that material costs and fixed costs dominate the cost per item produced, so that it is desirable to produce as many units as possible in a given time.

There are a variety of algorithms available for manipulator control. These algorithms usually assume that the control structure of the robot has been divided into two levels. The upper level is called *path* or *trajectory planning*, and the lower level is called *path control* or *path tracking*. Path control is the process of making the robots actual position and velocity match some desired values of position and velocity; the desired values are provided to the controller by the trajectory planner. The trajectory planner receives as input some sort of spatial path descriptor from which it calculates a time history of the desired positions and velocities. (The term "Path Planner" is often used in the literature; this is a misnomer, since it does not plan paths but rather supplies timing information to a pre-planned geometric curve. The term "trajectory planner" will therefore be used here.) The path tracker then compensates for any deviations of the actual position and velocity from the desired values.

The reason for dividing the control scheme in this way is that the process of robot control, if considered in its entirety, is very complicated, since the dynamics of all but the simplest robots are highly nonlinear and coupled. Dividing the controller into the two parts makes the whole process simpler. The path tracker is frequently a linear controller (e.g. a PID controller). While the nonlinearities of manipulator dynamics frequently are not taken into account at this level, such trackers can generally keep the manipulator fairly close to the desired trajectory. More sophisticated methods can be used, though, such as resolved motion rate control [33], resolved acceleration control [24], and various adaptive techniques [4, 6, 7, 14, 16, 17].

Unfortunately, the simplicity obtained from the division into trajectory planning and path tracking often comes at the expense of efficiency. The source of the



inefficiency is the trajectory planner. In order to use the robot efficiently, the trajectory planner must be aware of the robot's dynamic properties, and the more accurate the dynamic model is, the better the robot's capabilities can be used. However, most of the trajectory planning algorithms presented to date assume very little about the robot's dynamics. The usual assumption is that there are constant or piecewise constant bounds on the robots velocity and acceleration [19, 22, 23]. In fact, these bounds vary with position, payload mass, and even with payload shape. Thus in order to make the constant-upper-bound scheme work, the upper bounds must be chosen to be global greatest lower bounds of the velocity and acceleration limits; in other words, the worst case limits have to be used. Since the moments of inertia seen at the joints of the robot, and hence the acceleration limits, may vary by an order of magnitude as the robot moves from one position to another, such bounds can result in considerable inefficiency or under-utilization of the robot.

As previously mentioned, robot control is usually divided into trajectory planning and path tracking stages. At the lower (tracking) level, a great deal of work has been done, and a variety of methods have been used. One of the earliest tracking schemes was developed for a prosthetic arm by Whitney, and is called resolved-motion rate control [33]. Resolved motion rate control (RMRC) makes use of the manipulator Jacobian to transform desired velocities in some coordinate system which is natural for the task at hand (e.g. Cartesians) into velocities in joint coordinate space. The velocities are then sent to the individual joint servos. The scheme has the obvious advantage that it allows the robot to be driven in world, rather than joint, coordinates. However, generating the joint velocities requires the inversion of the manipulator Jacobian, which may be time-consuming, and may, if the manipula-

tor is at or near a degenerate configuration, be impossible or very inaccurate.

Luh, Walker and Paul have extended RMRC, adding resolution of accelerations as well as velocities [24]. The result is resolved-acceleration control, or RAC. The idea here is to generate desired joint accelerations from accelerations in world coordinates, and to use the Newton-Euler dynamics formulation to generate joint torques in real time. The accelerations, however, are still assumed to be "generated in some reasonable way"[24]. In order to utilize the robot fully, the trajectory planner must be aware of the robot's structure and actuator limits, and must take these factors into account. This is very difficult to do in world coordinates.

Another approach to path tracking is the use of adaptive controllers. Dubowsky and DesForges used the model-referenced adaptive control scheme (MRACS) to control a six degree-of-freedom manipulator at UCLA [7]. MRACS makes use of a reference model with the desired joint characteristics and a joint servo with adjustable feedback gains. The feedback gains are then adjusted on the fly so as to make the actual joint look like the reference model as much as possible. The results of MRACS appear to be quite good, but the system as presented in [7] is a simple position servo; this makes precise velocity control, as needed for minimum-time control schemes, difficult. It should be pointed out, though, that this is one of the few schemes where the effects of sampling on system stability and performance have been investigated [7], so that more is known about the theory of MRACS than about many other techniques.

Koivo and Guo used an auto-regressive discrete-time model for the manipulator dynamics, and minimized a quadratic error measure while estimating the coefficients of the time series with a Kalman filter [16, 17]. Chung and Lee use a Kalman filter

in a different way; they assume that the desired trajectory is given, generate nominal joint torques using the Newton-Euler method, and use a linear model to generate corrections to the a priori computed torques. They then use a Kalman filter to estimate the parameters for the linear perturbation model [4].

All the work described above either calculates joint actuator inputs blindly, i.e., does not take actuator limits into account explicitly, or assumes that accelerations have been given and computes actuator torques from the accelerations. But in order to drive the robot in an efficient way, the dynamics and actuator characteristics of the robot need to be taken into account. This rules out the use of methods such as RMRC. The computed torque methods, combined with an appropriate trajectory planner, provide one means of accomplishing this task, i.e., picking accelerations in a reasonable way.

One of the early minimum-time trajectory planning systems was developed by Luh and Walker [23]. It describes the desired manipulator path in terms of its initial and final points and a set of intermediate points. Each branch of the path has a maximum velocity assigned to it, and each intermediate point is assigned a maximum acceleration and a maximum position error. The time taken to go from the initial to the final point is, then, the sum of the times taken to traverse each branch of the path plus the sum of the times required to make the transitions from one branch to the next. The minimum possible sum of these times can then be found using linear programming. In this case, one must still choose appropriate maximum velocities and accelerations, and this cannot be done properly without either knowing the dynamic properties and actuator characteristics of the robot or having some experimental results which give maximum velocities and accelerations for given robot configura-

tions. Also, since maximum accelerations and velocities are assumed to be constant over some interval, it is necessary to choose them to be lower bounds of the maximum values over the given interval, i.e., worst case bounds on acceleration and velocity. Since these bounds will in general depend on position and velocity, this could result in under-utilization of the robot's capabilities.

Luh and Lin [22] present a modification of the scheme described above which uses nonlinear programming to generate the minimum-time trajectory. The major difference between the method of Luh and Lin and that of Luh and Walker is in a more careful treatment of the calculation of the times required for the transitions from one path segment to the next. Also, an efficient technique for solving the nonlinear programming problem is presented, along with a convergence proof.

Lin, Chang and Luh [19] present a third variation on this trajectory planning method. Instead of using path segments which are straight lines in Cartesian space, they convert points in Cartesian space into the equivalent joint coordinates, and pass a cubic spline through these points. The maximum velocities and accelerations are, then, joint velocities and accelerations, which are easier to compute from the robot dynamics and actuator characteristics. There is, however, still some calculation (or measurement) required in order to determine these quantities. This work, incidentally, also allows for limits on the jerk (time-derivative of acceleration). Placing limits on jerk helps prevent mechanism wear.

Kim and Shin [12, 13] have presented a method which is similar in some respects to the linear programming methods presented previously, but which uses the robot dynamic equations to obtain approximate acceleration bounds at each corner point. They also point out a set of conditions under which the linear programming

problem reduces to a set of local optimization problems, one for each corner point. This represents a change in computational complexity of from  $O(n^3)$  to  $O(n)$ .

Another type of trajectory planner has been developed by Bobrow, Dubowsky, and Gibson [3] and Shin and McKay [29, 30]. In these minimum-time trajectory planners, it is assumed that the desired path is given in a parameterized form. The parametric equations of the path can then be plugged into the manipulator dynamic equations, giving a set of second order differential equations in the (scalar) path parameter. Given these dynamic equations, bounds on individual joint torques can be converted into bounds on parametric accelerations (second time-derivatives of the path parameter); the allowable sets of second derivatives (one set per joint) are intersected, giving a single allowable set. Bounds on velocities (first derivatives of the path parameter) can also be found from these equations, since at some velocities there are no admissible accelerations. Then, using the fact that the minimum-time solution will be bang-bang in the acceleration, it is possible to construct phase plane plots which give the optimal trajectory in terms of the parameter and its derivatives. (The results found in [30] will be presented in Chapter 4.)

The papers [3] and [30] differ primarily in two respects: first, in the method of Bobrow et. al. the required search for trajectories is carried out by actually constructing trajectories and seeing where they go; in Shin and McKay, the search has been reduced to the problem of finding a sign change in an easily computable function. Second, some complications may arise with respect to computation of the admissible velocities for a given manipulator position. It is possible that there may be several distinct allowable velocity ranges for a given position. In [3] no mention is made about this possibility, and their search technique may fail if there are distinct

regions. Shin and McKay present a second algorithm to take care of this possibility.

While most people have taken the separate trajectory planning/trajectory tracking approach, several authors have made attempts at unified approaches to robot control. One early attempt was the near-minimum time control of Kahn and Roth [11], who linearized the dynamic equations, transformed the equations so as to eliminate coupling terms, and generated switching curves for this linear approximation. The result is a sub-optimal control which seems to give fairly good results if the initial and final states of the robot are fairly close. It does, however, have some problems with overshoot, as one would expect from such an approximation.

Another controller using an approximate dynamic model to generate optimal controls is the near-minimum time-fuel method of Kim and Shin [14]. They use a model which is linear over one sample period, and use coefficients in their linear model which result from averaging the coefficients at the current point on the trajectory and at the final point. The controls which result from this approximation depend upon whether time or fuel is the predominant term in the objective function, and upon the sampling rate. The trajectories for minimum fuel were slower than those for minimum time but had less overshoot. Increasing sampling rate also had the effect of reducing overshoot; thus here two parameters could be varied so as to find a good compromise between manipulator speed and overshoot.

One level up from trajectory planning is spatial planning, the process of finding collision-free paths for the manipulator to follow. Although the spatial planning problem is still largely unsolved, some work has been done, mostly with objects which are assumed to be spheres, cylinders, or convex polyhedra. Lozano-Perez [20] has reduced the spatial planning problem to two problems: find-space and find-path.

The find-space problem amounts to determining "safe" positions for the object being moved, i.e., positions where the object does not overlap any obstacles. Find-path is the process of finding a continuum of safe positions which takes the object from its initial configuration to a desired final configuration. In [20], Lozano-Perez presents algorithms for solving these problems in both two and three dimensions, though the three-dimensional algorithm does not in general give the "best" (minimum-distance) solution. In [21], Lozano-Perez describes the manipulator spatial planning problem in terms of volumes swept out by the links of the manipulator, and introduces polyhedral approximations of these swept volumes in order to use the results which are available for polyhedra.

Luh and Campbell [25] describe spatial planning in terms of obstacles and "pseudo-obstacles" which the manipulator must avoid. The pseudo-obstacles arise because all the links, and not just the payload, must avoid the real obstacles. Luh and Campbell determined the shapes of some of these pseudo-obstacles for the Stanford arm. In particular, they considered the problem of keeping the back end of the Stanford arm's single prismatic link from bumping into things, which has the effect of creating a pseudo-obstacle on the opposite side of the arm from the real obstacle which generates it. They also generated polyhedral approximations to these pseudo-obstacles.

More recently, Gilbert and Johnson [8] have developed a technique for determining optimal controls in the presence of obstacles. Their technique starts with some feasible path, given as a cubic spline. They then apply a gradient technique which both moves the points interpolated by the spline and changes the robot's velocity in such a way as to optimize some performance measure. Obstacle avoidance is

accomplished by including penalty functions in the performance index.

Another aspect of the trajectory planning problem is the actual generation of path planners. For those methods which require the manipulator dynamic equations, this means that the equations must be derived. Since this process involves large amounts of symbolic calculation and is therefore highly susceptible to human error, several authors have attempted to mechanize the process of deriving the manipulator dynamic equations. In particular, Bejczy and Paul [1] and Luh and Lin [26] have done work in this area. Bejczy and Paul describe methods for determining when certain dynamic coefficients must be zero for geometric reasons or because of symmetry. Luh and Lin describe a method for manipulating the dynamic coefficients symbolically, and give criteria for eliminating insignificant terms in the dynamic equations.

This thesis assumes that the geometric path planner has generated a parameterized curve in joint space, as described in [3] and [30]. The trajectory planning problem can then be reduced to a problem of small dimension by converting all the dynamic and actuator constraints to constraints on the single parameter which is used to describe the path, and the parameter's time derivatives. Within this framework, a variety of optimization techniques can be applied. The optimization methods described here apply to both minimum time problems and to more general minimum cost control problems. It is also possible to modify the constraints to take uncertain dynamics into account at the trajectory planning stage.

The remainder of this thesis is structured as follows: Chapter 2 introduces the mathematical notation used throughout the following chapters, and gives a derivation of the dynamic model of a manipulator. Chapter 3 formally states the problems to be solved. Chapter 4 presents three related but distinct trajectory planning



algorithms, and gives results of these algorithms as applied to either a simple two degree-of-freedom arm or the first three joints of the Bendix PACS robot arm. Chapter 5 gives some results on selection of near-optimal (in the minimum-time sense) geometric paths. Chapter 6 presents a modification of one of the trajectory planners described in Chapter 4 which allows for uncertainties in the dynamic characteristics of the manipulator which is being controlled. Chapter 7 discusses how trajectory planners can be generated automatically. The thesis concludes with Chapter 8.

## CHAPTER 2

### MATHEMATICAL PRELIMINARIES

Much of the discussion in subsequent chapters involves the dynamic equations of robot arms, and makes use of tensor analysis and Riemannian geometry. This brief introduction should help those who are familiar with the ideas of vector and matrix notation and elementary differential geometry but are unfamiliar with tensor analysis.

#### 2.1. Tensors and Tensor Notation

Tensor notation is much like vector notation except that the symbols used in tensor equations may have subscripts and/or superscripts. A vector is written as a symbol with one index, and a matrix will have two. It is possible in tensor notation to write arrays of three or more dimensions; a three-dimensional array simply has three indices. Tensor notation also provides some useful tools for dealing with curves in spaces of an arbitrary number of dimensions.

Formally, a tensor is a quantity or set of quantities which obeys certain rules when transformed from one set of curvilinear coordinates to another. The transformation rules are of two types, as indicated by the position of the tensor's indices. Superscripts indicate that the index is *contravariant* and subscripts indicate that it is

*covariant*. A contravariant tensor of order one, or a contravariant vector, transforms from one set of curvilinear coordinates to another (in this case, from unprimed to primed coordinate systems) according to the law

$$\mathbf{T}'^i = \sum_{j=1}^N \frac{\partial q'^i}{\partial q^j} \mathbf{T}^j \quad (2.1.1)$$

where  $N$  is the dimension of the space under consideration and the  $q^i$  and  $q'^j$  are the coordinates in the unprimed and primed coordinate systems [32]. Likewise, a covariant vector transforms according to the rule [32]

$$\mathbf{T}'_i = \sum_{j=1}^N \frac{\partial q^j}{\partial q'^i} \mathbf{T}_j \quad (2.1.2)$$

Note that the differentials of the coordinates,  $dq^i$ , transform according to the contravariant rule, though in general the coordinates themselves do not. Writing out the contravariant transformation rule with  $dq^j$  written for  $\mathbf{T}^j$  and  $dq'^j$  written for  $\mathbf{T}'^j$  and dividing by  $dt$  shows that velocities are contravariant vectors.

A single quantity which retains its value when transformed from one coordinate system to another is called an *invariant* or a *scalar*. It is easily verified that the partial derivatives of an invariant with respect to the coordinates transform according to the covariant rule.

The generalization to tensors of higher order is straightforward. Tensors of order two can have two subscripts, two superscripts, or one subscript and one superscript. These are called covariant, contravariant, and mixed tensors of order two, and they transform according to the laws

$$\mathbf{T}'_{ij} = \sum_{r=1}^N \sum_{s=1}^N \frac{\partial q^r}{\partial q'^i} \frac{\partial q^s}{\partial q'^j} \mathbf{T}_{rs} \quad (2.1.3)$$

$$\mathbf{T}'^{ij} = \sum_{r=1}^N \sum_{s=1}^N \frac{\partial q'^i}{\partial q^r} \frac{\partial q'^j}{\partial q^s} \mathbf{T}^{rs} \quad (2.1.4)$$

$$\mathbf{T}'_{,j} = \sum_{r=1}^N \sum_{s=1}^N \frac{\partial q^r}{\partial q'^i} \frac{\partial q'^j}{\partial q^s} \mathbf{T}_{,j} \quad (2.1.5)$$

The generalization to tensors of order three and higher follows the obvious pattern.

An important notational tool is the so-called *summation convention*. If an index appears twice in a product of two tensor expressions, then the expression is summed from 1 to  $N$  over the repeated index. Thus  $\mathbf{a}, \mathbf{b}^i$  is shorthand for  $\sum_{i=1}^N \mathbf{a}, \mathbf{b}^i$ .

Using this shorthand, the transformation rules for tensors of orders one and two are written as

$$\mathbf{T}'^i = \frac{\partial q'^i}{\partial q^j} \mathbf{T}^j \quad \mathbf{T}'_i = \frac{\partial q^j}{\partial q'^i} \mathbf{T}_j \quad (2.1.6)$$

$$\mathbf{T}'_{,ij} = \frac{\partial q^r}{\partial q'^i} \frac{\partial q^s}{\partial q'^j} \mathbf{T}_{,rs} \quad \mathbf{T}'^{,ij} = \frac{\partial q'^i}{\partial q^r} \frac{\partial q'^j}{\partial q^s} \mathbf{T}^{,rs} \quad (2.1.7)$$

$$\mathbf{T}'_{,j} = \frac{\partial q^r}{\partial q'^i} \frac{\partial q'^j}{\partial q^s} \mathbf{T}_{,j} \quad (2.1.8)$$

It is important to note that a given index should not appear more than twice in any term of a tensor equation, and that repeated indices should appear once as a subscript and once as a superscript. If a repeated index appears, for example, twice as a subscript, then the resulting quantity will not in general be a tensor. If, on the other

hand, the index appears once as a subscript and once as a superscript, it is easily verified that the expression is a tensor whose character is indicated by the remaining (non-repeated) indices. For example,  $T_{ij}x^j$  is a covariant tensor of order one; the  $j$  indices "cancel".

## 2.2. Riemannian Spaces and Properties of Curves

Another advantage of tensor notation is that it is also the language of Riemannian geometry, so that the tools of this field of mathematics are available. As the reader may be aware, Riemannian geometry consists of the study of the metrical properties of spaces of an arbitrary number of dimensions. The space is described by its *metric tensor*, which gives the square of the differential line element as a quadratic form in the differentials of the coordinates, i.e.,

$$ds^2 = J_{ij} dq^i dq^j \quad (2.2.1)$$

where  $J_{ij}$  is the metric tensor and the  $q^i$  are the coordinates.  $J_{ij}$  may without loss of generality be assumed to be symmetric, so that  $J_{ij} = J_{ji}$ . It also will be assumed throughout this work that the metric tensor is positive definite, i.e., that  $J_{ij}x^ix^j > 0$  for all  $x \neq 0$ . Geometries in which this is not true can be developed, and indeed are used in general relativity theory, but for the cases dealt with here,  $J_{ij}$  will always be positive definite.

The introduction of the metric tensor allows distances and angles to be measured, and allows the computation of norms of vectors. Distances along curves are calculated by integrating the formula for the differential line element  $ds$ . The norm of a contravariant vector is given by the formula

$$\| \mathbf{x}^i \|^2 \equiv \mathbf{J}_{ij} \mathbf{x}^i \mathbf{x}^j \quad (2.2.2)$$

The angle between two vectors is given by the formula

$$\cos \theta = \frac{\mathbf{J}_{ij} \mathbf{x}^i \mathbf{y}^j}{\| \mathbf{x} \| \cdot \| \mathbf{y} \|} \quad (2.2.3)$$

If  $\mathbf{J}_{ij}$  is positive definite, then it can be shown that the righthand side of equation (2.2.3) is always between +1 and -1, so that  $\theta$  is a real angle. Even if  $\mathbf{J}$  is not positive definite, two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are said to be orthogonal if and only if  $\mathbf{J}_{ij} \mathbf{x}^i \mathbf{y}^j = 0$ .

Note that the expression  $\mathbf{J}_{ij} \mathbf{x}^i \mathbf{y}^j$  behaves much as the scalar product behaves in Euclidean space, and indeed reduces to the scalar product in Euclidean space with Cartesian coordinates.

The curves in a Riemannian space corresponding to straight lines are *geodesics*, or curves of minimum distance. The differential equation which describes these curves can be found from the form of the line element  $ds$  using variational techniques. Using such techniques, the differential equation obtained is

$$\mathbf{J}_{ij} \frac{d^2 \mathbf{q}^j}{ds^2} + [jk, i] \frac{d \mathbf{q}^j}{ds} \frac{d \mathbf{q}^k}{ds} = 0 \quad (2.2.4)$$

The symbol  $[jk, i]$  is a *Christoffel symbol of the first kind*, and is defined by

$$[jk, i] \equiv \frac{1}{2} \left( \frac{\partial J_{ij}}{\partial \mathbf{q}^k} + \frac{\partial J_{ik}}{\partial \mathbf{q}^j} + \frac{\partial J_{jk}}{\partial \mathbf{q}^i} \right) \quad (2.2.5)$$

It should be noted that these symbols are *not* tensors.

Since the metric tensor  $\mathbf{J}_{ij}$  is positive definite, it is also invertible. The inverse of this matrix is denoted by  $\mathbf{J}^{jk}$ , and we have the relationship

$$J_{ij} J^{jk} = \delta_i^k = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases} \quad (2.2.6)$$

Multiplying a contravariant tensor  $\mathbf{x}^j$  by  $J_{ij}$  and summing over  $j$  gives a new covariant tensor  $\mathbf{x}_i \equiv J_{ij} \mathbf{x}^j$ . This operation is called *lowering a suffix*. Likewise, a suffix can be raised by multiplying by  $J^{ij}$  and summing. Thus  $\mathbf{x}^i = J^{ij} \mathbf{x}_j$ .

If equation (2.2.4) is multiplied by  $J^{mi}$  and summed over  $i$  we obtain the equation

$$\frac{d^2 \mathbf{q}^m}{ds^2} + \left\{ \begin{matrix} m \\ jk \end{matrix} \right\} \frac{d \mathbf{q}^j}{ds} \frac{d \mathbf{q}^k}{ds} = 0 \quad (2.2.7)$$

The symbol  $\left\{ \begin{matrix} m \\ jk \end{matrix} \right\}$  is the *Christoffel symbol of the second kind*, and is defined as

$$\left\{ \begin{matrix} m \\ jk \end{matrix} \right\} \equiv J^{mi} [jk, i] \quad (2.2.8)$$

As a special case, consider ordinary euclidean space with rectangular cartesian coordinates. Then the metric tensor is just the identity matrix (or Kronecker delta)  $\delta_{ij}$ ; and the Christoffel symbols are zero. Reassuringly, equations (2.2.4) and (2.2.7) then reduce to  $\frac{d^2 \mathbf{q}^m}{ds^2} = 0$ , the differential equations which describe straight lines.

Equation (2.2.7) is often written

$$\frac{\delta}{\delta s} \left( \frac{d \mathbf{q}^i}{ds} \right) = 0 \quad (2.2.9)$$

where the operator  $\frac{\delta}{\delta s}$  is called the *absolute derivative* with respect to  $s$ . The absolute derivative of a contravariant tensor of order one with respect to the scalar  $\phi$  is defined as

$$\frac{\delta \mathbf{T}^i}{\delta \phi} = \frac{d \mathbf{T}^i}{d \phi} + \left\{ \begin{matrix} i \\ jk \end{matrix} \right\} \mathbf{T}^j \frac{d q^k}{d \phi} \quad (2.2.10)$$

Likewise, the absolute derivative of a covariant tensor is

$$\frac{\delta \mathbf{T}_i}{\delta \phi} = \frac{d \mathbf{T}_i}{d \phi} - \left\{ \begin{matrix} j \\ ik \end{matrix} \right\} \mathbf{T}_j \frac{d q^k}{d \phi} \quad (2.2.11)$$

Each of these derivatives consists of the ordinary derivative followed by a term involving a Christoffel symbol. The absolute derivative of a tensor of order two has two terms involving Christoffel symbols, tensors of order three have three additional terms, and so forth. The absolute derivative of an invariant is just the invariant's ordinary derivative. It can be shown that the absolute derivative of a tensor is itself a tensor, unlike the ordinary derivative; this property makes the absolute derivative a useful quantity.

In many ways the absolute derivative behaves as an ordinary derivative behaves; it obeys the same rules for derivatives of sums and products, and obeys something like the chain rule. In particular

$$\frac{\delta}{\delta \phi} (\mathbf{A}^i + \mathbf{B}^i) = \frac{\delta \mathbf{A}^i}{\delta \phi} + \frac{\delta \mathbf{B}^i}{\delta \phi} \quad (2.2.12)$$

$$\frac{\delta}{\delta \phi} (\mathbf{A}^i \mathbf{B}_i) = \frac{\delta \mathbf{A}^i}{\delta \phi} \mathbf{B}_i + \mathbf{A}_i \frac{\delta \mathbf{B}^i}{\delta \phi} \quad (2.2.13)$$

$$\frac{\delta}{\delta \phi} (\mathbf{A}^i \mathbf{B}_j) = \frac{\delta \mathbf{A}^i}{\delta \phi} \mathbf{B}_j + \mathbf{A}_i \frac{\delta \mathbf{B}^j}{\delta \phi} \quad (2.2.14)$$

$$\frac{\delta \mathbf{A}^i}{\delta \phi} = \frac{\delta \mathbf{A}^i}{\delta \psi} \frac{d \psi}{d \phi} \quad (2.2.15)$$



A curve in a Riemannian space can be written parametrically as

$$\mathbf{q}^i = f^i(\lambda) \quad (2.2.16)$$

where  $\lambda$  is some scalar parameter. The derivative of this position vector,  $\frac{d\mathbf{q}^i}{d\lambda}$ , is the *tangent* to the curve. In particular, if  $\lambda$  is the line element  $s$  then it is the *unit tangent* to the curve. The absolute derivative of the unit tangent,  $\frac{\delta}{\delta s} \left( \frac{d\mathbf{q}^i}{ds} \right)$ , is the *curvature vector*. Note that for a geodesic the curvature vector is zero.

As in ordinary differential geometry, the curvature vector is orthogonal to the unit tangent. To see this, consider the identity

$$1 = \mathbf{J}_{ij} \mathbf{p}^i \mathbf{p}^j \quad (2.2.17)$$

where  $\mathbf{p}^i$  is the unit tangent  $\frac{d\mathbf{q}^i}{ds}$ . Taking the absolute derivative of both sides of the equation,

$$0 = \frac{\delta \mathbf{J}_{ij}}{\delta s} \mathbf{p}^i \mathbf{p}^j + \mathbf{J}_{ij} \frac{\delta \mathbf{p}^i}{\delta s} \mathbf{p}^j + \mathbf{J}_{ij} \mathbf{p}^i \frac{\delta \mathbf{p}^j}{\delta s} \quad (2.2.18)$$

It can be shown that the absolute derivative of the metric tensor is identically zero, so

$$0 = \mathbf{J}_{ij} \frac{\delta \mathbf{p}^i}{\delta s} \mathbf{p}^j + \mathbf{J}_{ij} \mathbf{p}^i \frac{\delta \mathbf{p}^j}{\delta s} = 2\mathbf{J}_{ij} \mathbf{p}^i \frac{\delta \mathbf{p}^j}{\delta s} \quad (2.2.19)$$

Thus the unit tangent and the curvature vector are orthogonal.

### 2.3. Robot Arm Dynamics

In order to control a robot properly, the robot's dynamics must be known. There are a number of ways of obtaining a robot's dynamic equations, the two most

commonly used methods being Lagrange's method [9, 28] and the Newton-Euler method [24]. The Newton-Euler formulation gives a set of recursive equations. These recursive equations require relatively few numerical calculations, but are not well suited to use in control problems. Lagrange's equations, on the other hand, express the joint torques/forces in terms of differential equations. Most commonly, these equations are written non-recursively, though recursive formulations do exist [31]. For the problems to be dealt with here, the (non-recursive) Lagrangian form is the easiest form to use.

#### 2.4. Derivation of the Lagrangian Dynamic Equations

The Lagrangian equations of motion are [9, 28]

$$u_i = \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}^i} \right) - \frac{\partial L}{\partial q^i} \quad (2.4.1)$$

where  $u_i$  is the  $i^{\text{th}}$  generalized force,  $q^i$  is the  $i^{\text{th}}$  generalized coordinate, and the Lagrangian  $L = K - P =$  kinetic energy minus potential energy. For a robot arm, or any other system of mechanical linkages for that matter, the kinetic energy  $K$  has the form

$$K = \frac{1}{2} J_{jk}(\mathbf{q}) \dot{q}^j \dot{q}^k \quad (2.4.2)$$

where  $J_{jk}$  is a symmetric, positive-definite matrix describing the inertia of the robot and the inertial coupling between the robot's joints. (See [28] for a method of obtaining this matrix.) The Einstein summation convention has been used here, as described in Section 2.1. The index values range from 1 to  $N$ , where  $N$  is the number of joints the robot has.

The potential energy  $P$  is a function of gravity and of the position of the robot arm, so that we can take  $P = P(\mathbf{q})$ . The value of  $L$  is then

$$L = \frac{1}{2} \mathbf{J}_{jk}(\mathbf{q}) \dot{q}^j \dot{q}^k - P(\mathbf{q}) \quad (2.4.3)$$

Computing the partial derivative  $\partial L / \partial \dot{q}^i$ ,

$$\frac{\partial L}{\partial \dot{q}^i} = \frac{1}{2} \mathbf{J}_{ik}(\mathbf{q}) \dot{q}^k + \frac{1}{2} \mathbf{J}_{ji}(\mathbf{q}) \dot{q}^j \quad (2.4.4)$$

Making use of the symmetry of  $\mathbf{J}$  and changing dummy indices,

$$\frac{\partial L}{\partial \dot{q}^i} = \frac{1}{2} \mathbf{J}_{ij}(\mathbf{q}) \dot{q}^j + \frac{1}{2} \mathbf{J}_{ij}(\mathbf{q}) \dot{q}^j = \mathbf{J}_{ij}(\mathbf{q}) \dot{q}^j \quad (2.4.5)$$

Differentiating this with respect to time gives

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}^i} \right) = \mathbf{J}_{ij}(\mathbf{q}) \ddot{q}^j + \frac{\partial \mathbf{J}_{ij}(\mathbf{q})}{\partial q^k} \dot{q}^j \dot{q}^k \quad (2.4.6)$$

Differentiating  $L$  with respect to  $q^k$ ,

$$\frac{\partial L}{\partial q^k} = \frac{1}{2} \frac{\partial \mathbf{J}_{jk}(\mathbf{q})}{\partial q^i} \dot{q}^j \dot{q}^k - \frac{\partial P(\mathbf{q})}{\partial q^k} \quad (2.4.7)$$

Plugging the results from (2.4.6) and (2.4.7) into (2.4.1) gives

$$u_i = \mathbf{J}_{ij}(\mathbf{q}) \dot{q}^j + \left( \frac{\partial \mathbf{J}_{ij}(\mathbf{q})}{\partial q^k} - \frac{1}{2} \frac{\partial \mathbf{J}_{jk}(\mathbf{q})}{\partial q^i} \right) \dot{q}^j \dot{q}^k + \frac{\partial P(\mathbf{q})}{\partial q^i} \quad (2.4.8)$$

Using the symmetry of  $\mathbf{J}_{ij}$ , a simple manipulation of dummy indices shows that the term

$$\left( \frac{\partial \mathbf{J}_{ij}(\mathbf{q})}{\partial \mathbf{q}^k} - \frac{1}{2} \frac{\partial \mathbf{J}_{jk}(\mathbf{q})}{\partial \mathbf{q}^i} \right) \dot{\mathbf{q}}^j \dot{\mathbf{q}}^k \quad (2.4.9)$$

may be written as

$$\frac{1}{2} \left( \frac{\partial \mathbf{J}_{ij}(\mathbf{q})}{\partial \mathbf{q}^k} + \frac{\partial \mathbf{J}_{ik}(\mathbf{q})}{\partial \mathbf{q}^j} - \frac{\partial \mathbf{J}_{jk}(\mathbf{q})}{\partial \mathbf{q}^i} \right) \dot{\mathbf{q}}^j \dot{\mathbf{q}}^k = [jk, i] \dot{\mathbf{q}}^j \dot{\mathbf{q}}^k \quad (2.4.10)$$

The square bracket symbol is the Christoffel symbol of the first kind computed with respect to the metric  $\mathbf{J}_{ij}$ .

The dynamic equations of an  $N$ -degree-of-freedom robot arm thus take the general form

$$\dot{\mathbf{q}}^i = \mathbf{v}^i \quad (2.4.11)$$

$$\mathbf{u}_i = \mathbf{J}_{ij}(\mathbf{q}) \dot{\mathbf{v}}^j + \mathbf{R}_{ij} \mathbf{v}^j + \mathbf{C}_{ijk}(\mathbf{q}) \mathbf{v}^j \mathbf{v}^k + \mathbf{g}_i(\mathbf{q}) \quad (2.4.12)$$

where

$\mathbf{q}^i = i^{\text{th}}$  generalized coordinate

$\mathbf{v}^i = i^{\text{th}}$  generalized velocity

$\mathbf{u}_i = i^{\text{th}}$  generalized force

$\mathbf{J}_{ij} =$  the inertia matrix

$\mathbf{g}_i = \frac{\partial P}{\partial \mathbf{q}^i} =$  gravitational force on the  $i^{\text{th}}$  joint

$\mathbf{C}_{ijk} = [jk, i] =$  array of Coriolis and centrifugal coefficients

and

$\mathbf{R}_{ij} =$  viscous friction matrix.

Note the correction for frictional effects, i.e., the introduction of the term involving  $\mathbf{R}_{ij}$ .

As an alternative, the definition of the absolute derivative, equation (2.2.10), may be used to rewrite equation (2.4.12) as

$$u_i = J_{ij} \frac{\delta v^j}{\delta t} + R_{ij} v^j + g_i(\mathbf{q}) \quad (2.4.13)$$

This equation is a tensor equation. To see this, consider each term on the right side of (2.4.12) individually.  $J_{ij}$  is known to be a covariant tensor of order two, since the (invariant) kinetic energy is given by  $J_{ij} v^i v^j$  for any arbitrary velocity  $v^i$ . Since  $J^{ij}$  is symmetric, this proves that  $J_{ij}$  has the tensor character indicated [32]. Since  $v^i$  is a tensor, so is  $\frac{\delta v^j}{\delta t}$ , and hence so is  $J_{ij} \frac{\delta v^j}{\delta t}$ .

Likewise, the power consumed by frictional effects is an invariant, and is given by  $R_{ij} v^i v^j$ . By an argument similar to that used to prove that  $J_{ij}$  is a tensor,  $R_{ij}$  must also be a tensor, and hence also  $R_{ij} v^j$ . The gravitational term  $g_i$  is the partial derivative of the potential energy, which is of course invariant, so it is a covariant tensor of order one, as indicated by its single subscript.

Since the right-hand side of equation (2.4.12) is a tensor, the left-hand side must also be a tensor. Hence the equation (2.4.12) is in tensor form. This implies that (2.4.12) will have the same form in any system of curvilinear coordinates, and all the quantities in the equation transform according to the tensor transformation laws given in the previous chapter. (Note, however, that these transformation laws break down if the determinant of the Jacobian matrix,  $\frac{\partial q'^i}{\partial q^j}$ , is zero.)

In Chapter 5 it will prove useful to express the dynamic equations (2.4.13) in terms of absolute derivatives with respect to arc length  $s$ . For our purposes, we will define arc length  $ds$  by the quadratic form  $ds^2 \equiv J_{ij} dq^i dq^j$ . Since the kinetic energy of the manipulator is given by

$$K = \frac{1}{2} J_{ij} \frac{dq^i}{dt} \frac{dq^j}{dt}, \quad (2.4.14)$$

it can be seen that the infinitesimal arc  $ds$  in this space is related to the kinetic energy of the manipulator by the formula  $\left(\frac{ds}{dt}\right)^2 = 2K$ .

The dynamic equations may now be expressed in terms of the arc length  $s$  and the time derivatives of  $s$ . We have, since the absolute derivative obeys the chain rule,

$$u_i = J_{ij} \frac{\delta v^j}{\delta s} \frac{ds}{dt} + R_{ij} v^j + g_i. \quad (2.4.15)$$

Using the relationship  $v^j = \frac{dq^j}{ds} \frac{ds}{dt}$ , then

$$u_i = J_{ij} \frac{\delta}{\delta s} \left( p^j \frac{ds}{dt} \right) \frac{ds}{dt} + R_{ij} p^j \frac{ds}{dt} + g_i \quad (2.4.16)$$

where  $p^j \equiv \frac{dq^j}{ds}$  is the unit tangent to the manipulator's path. But

$$\begin{aligned} \frac{\delta}{\delta s} \left( p^j \frac{ds}{dt} \right) &= \frac{\delta p^j}{\delta s} \frac{ds}{dt} + p^j \frac{\delta}{\delta s} \left( \frac{ds}{dt} \right) \\ &= \frac{\delta p^j}{\delta s} \frac{ds}{dt} + p^j \frac{d}{ds} \left( \frac{ds}{dt} \right) \end{aligned} \quad (2.4.17)$$

since the absolute derivative of a scalar is just the ordinary derivative. Plugging this into the dynamic equations (2.4.16),

$$\mathbf{u}_i = \mathbf{J}_{ij} \frac{\delta \mathbf{p}^j}{\delta s} \left( \frac{ds}{dt} \right)^2 + \mathbf{J}_{ij} \mathbf{p}^j \frac{ds}{dt} \frac{d}{ds} \left( \frac{ds}{dt} \right) + \mathbf{R}_{ij} \mathbf{p}^j \frac{ds}{dt} + \mathbf{g}_i. \quad (2.4.18)$$

Using the identity  $\frac{ds}{dt} \frac{d}{ds} \left( \frac{ds}{dt} \right) \equiv \frac{d}{dt} \left( \frac{ds}{dt} \right) \equiv \frac{d^2 s}{dt^2}$ ,

$$\mathbf{u}_i = \mathbf{J}_{ij} \frac{\delta \mathbf{p}^j}{\delta s} \left( \frac{ds}{dt} \right)^2 + \mathbf{J}_{ij} \mathbf{p}^j \frac{d^2 s}{dt^2} + \mathbf{R}_{ij} \mathbf{p}^j \frac{ds}{dt} + \mathbf{g}_i. \quad (2.4.19)$$

It is interesting to consider the form of the last equation. The left-hand side consists of externally applied forces. There are four terms on the right-hand side: a term proportional to the square of the velocity, a term proportional to the acceleration, a viscous friction term which is proportional to the velocity, and a gravitational term which is a function only of position. The first two terms are of particular interest. They are just the Coriolis and tangential acceleration terms respectively. The Coriolis term is just the (vector) curvature of the path multiplied by the square of the speed, and so has a form analogous to the familiar  $\frac{mv^2}{r}$  term encountered in uniform circular motion. The second term, likewise, looks like the classical  $ma$  term one sees in one-dimensional Newtonian mechanics. The most important fact to note is that it is clear from this form of the dynamic equations that the Coriolis terms result directly from the curvature of the path in the manipulator's inertia space.

The work  $W$  done on the manipulator is

$$W = \int \mathbf{u}_i dq^i = \int \mathbf{u}_i \frac{dq^i}{ds} ds = \int \mathbf{u}_i \mathbf{p}^i ds. \quad (2.4.20)$$

Plugging in the expression for  $\mathbf{u}_i$  from Eq. (2.4.18),

$$\begin{aligned} W = \int \left[ \mathbf{J}_{ij} \mathbf{p}^i \frac{\delta \mathbf{p}^j}{\delta s} \frac{ds}{dt} + \mathbf{J}_{ij} \mathbf{p}^i \mathbf{p}^j \frac{d}{ds} \left( \frac{ds}{dt} \right) \right] \frac{ds}{dt} ds \\ + \int \mathbf{R}_{ij} \mathbf{p}^i \mathbf{p}^j \frac{ds}{dt} ds + \int \mathbf{g}_i \mathbf{p}^i ds. \end{aligned} \quad (2.4.21)$$

Using the facts that the curvature vector  $\frac{\delta \mathbf{p}^j}{\delta s}$  is orthogonal to the unit tangent  $\mathbf{p}^j$  and that  $\mathbf{p}^i$  is a unit vector, i.e., that  $\mathbf{J}_{ij} \mathbf{p}^i \mathbf{p}^j = 1$ , Eq. (2.4.21) transforms to

$$\begin{aligned} W = \int \left( \frac{ds}{dt} \right) \frac{d}{ds} \left( \frac{ds}{dt} \right) ds + \int \mathbf{R}_{ij} \mathbf{p}^i \mathbf{p}^j \frac{ds}{dt} ds + \int \mathbf{g}_i \mathbf{p}^i ds \\ = \frac{1}{2} \left( \frac{ds}{dt} \right)^2 + \int \mathbf{R}_{ij} \mathbf{p}^i \mathbf{p}^j \frac{ds}{dt} ds + \int \mathbf{g}_i \mathbf{p}^i ds. \end{aligned} \quad (2.4.22)$$

The power consumed by the manipulator at any given time is just

$$P \equiv \frac{dW}{dt} = \frac{ds}{dt} \frac{d^2s}{dt^2} + \mathbf{R}_{ij} \mathbf{p}^i \mathbf{p}^j \left( \frac{ds}{dt} \right)^2 + \mathbf{g}_i \mathbf{p}^i \frac{ds}{dt}. \quad (2.4.23)$$



## CHAPTER 3

### PROBLEM STATEMENT

The goal of automation, as previously stated, is to produce goods at as low a cost as possible. In practice, costs may be divided into two groups: fixed and variable. Variable costs depend upon details of the manufacturing process, and include, in the cases where robots are used, that part of the cost of driving a robot which varies with robot motion, and some maintenance costs. Fixed costs are those which remain constant on a per-unit-time basis. Fixed costs include taxes, heating costs, building maintenance, and, in the case of a robot, the portion of the electric power which the robot uses to run its computer controller and other peripheral devices. If one assumes that the fixed costs dominate, then cost per item produced will be proportional to the time taken to produce the item. In other words, minimum cost is equivalent to minimum production time. This important special case will be treated in some detail later. A loose statement of the minimum-cost path planning problem is as follows:

What control signals will drive a given robot from a given initial configuration to a given final configuration with as small a cost as possible, given constraints on the magnitudes of the control signals and constraints on the intermediate configurations of the robot, i.e., given that the robot must not hit any obstacles?

While the problem of avoiding obstacles in the robot's workspace is not easily formulated as a control theory problem, the problem of moving a mechanical system with minimum cost is. One way to sidestep the collision avoidance problem, then, is to assume that the desired path has been specified a priori, for example as a parameterized curve in the robot's joint space. If this assumption is added, then one obtains a second, slightly different problem statement:

What controls will drive a given robot along a specified curve in joint space with minimum cost, given constraints on initial and final velocities and on control signal magnitudes?

This form of the problem reduces the complexity of the control problem by introducing a single parameter which describes the robot's position. The time derivative of this parameter and the parameter itself completely describe the current state (joint positions and velocities) of the robot. The control problem then becomes essentially a two dimensional minimum-cost control problem with some state and input constraints.

In this thesis, the minimum-cost control problem will be divided into two sub-problems:

1. Given a curve in the robot's joint space (or some equivalent coordinate system), the robot's dynamic properties, and the robot's actuator characteristics, what set of signals to the actuators will drive the robot from its current state to a desired final state with minimum cost?
2. Given the solution to problem 1, what curve should be selected, i.e., what curve will give the best (minimum-cost) solution?

This thesis will present several methods for solving the first problem, and some approximate methods for solving the second.

To state problem 1 more formally, assume that the geometric path is given in the form of a parameterized curve, say

$$\mathbf{q}^i = f^i(\lambda), \quad 0 \leq \lambda \leq \lambda_{\max} \quad (3.1)$$

where  $\mathbf{q}^i$  is the position of the  $i^{\text{th}}$  joint, and the initial and final points on the trajectory correspond to the points  $\lambda=0$  and  $\lambda=\lambda_{\max}$ . Also assume that the bounds on the actuator torques can be expressed in terms of the state of the system, i.e., in terms of the robot's speed and position, so that

$$\mathbf{u} \in \mathbf{E}(\mathbf{q}, \dot{\mathbf{q}}) \quad (3.2)$$

where  $\mathbf{u}$  is a vector of actuator torques/forces, and  $\mathbf{E}: R^N \times R^N \rightarrow R^N$  is a set function.  $N$  is the number of joints the robot has. Given the functions  $f^i$ , the set function  $\mathbf{E}$ , the desired initial and final velocities, and the manipulator dynamic equations (2.4.11) and (2.4.12), problem 1 is to find the controls  $\mathbf{u}(\lambda)$  which minimize the cost functional  $C$  given by

$$C = \int_0^{\lambda_{\max}} \phi(\mathbf{u}(\lambda), \mathbf{q}(\lambda), \dot{\mathbf{q}}(\lambda)) d\lambda \quad (3.3)$$

Problem 2 may be stated in much the same way as problem 1, except that one set of constraints, the set of geometric path constraints, is replaced by a more general set of constraints. These more general constraints state that the robot must not pass through any configuration in which it hits obstacles in its workspace. There will in general be an infinite number of ways in which the robot can do this, even when many obstacles must be avoided in the workspace. In the past, paths have frequently been constructed simply by connecting a set of corner points with straight

lines and then allowing the path planner to “round off” the corners. But straight lines are not simple motions to produce for most robots, and so are probably not the best paths to choose. It should also be noted that the shortest path may not be the minimum-cost path. In particular, the shortest cartesian path may have one or more corners, and the robot would have to come to a complete stop at these points. If the objective is, for example, to minimize traversal time, then one would almost certainly not want a path with any corners in it.

There are several other problems which must be surmounted in order to make a path planning scheme practical. In particular, in minimum-time path planning, one or more joints are driven at maximum torque. If there are any errors in the computed torques, such as those caused by modeling errors, then the path planner may ask the path tracker to use a higher torque than it is capable of generating. The robot will then stray from the desired path. It is therefore necessary to carry out path planning calculations with this in mind, so that any variation in the parameter measurements can be accommodated by the path tracker without saturating the actuators. In other words, if the robot's nominal dynamic equations are given by

$$\mathbf{u}_i = \mathbf{F}_1(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \quad (3.4)$$

and its actual dynamic equations are given by

$$\mathbf{u}_i = \mathbf{F}_2(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \quad (3.5)$$

then it is desired that changes in the required torque due to the error  $\mathbf{F}_2 - \mathbf{F}_1$  not result in a torque request which exceeds the capacity of the actuators. This leads to the problem

3. Given bounds on the modeling errors for the robot's dynamics, how can the trajectory planning scheme be altered so that the modelling errors do not make the actual required torques exceed the torques which the actuators can provide?

Another practical aspect of the trajectory planning problem is that of the description of curves and the actual calculation of actuator torques. Some suitable method of representing curves is required, and all computations involving those curves should be done automatically. In particular, it should be possible, given a robot's dynamic equations, to generate a path planner for that robot. This is especially desirable in view of the fact that the dynamic equations of all but the simplest robots are very complicated, and any manipulation of such equations will be prone to human error. This gives rise to another problem, namely

4. Given a robot's dynamic equations, is it practical to generate a path planner for that robot automatically?

In summary, the robot path planning problem can be divided into four parts:

1. Trajectory planning: Given a geometric path in joint space expressed as a parameterized curve, actuator torque limits expressed in terms of the robot's position and velocity, the dynamic equations of the robot, desired initial and final velocities, and (possibly) externally-imposed constraints on velocity and jerk (the time-derivative of acceleration), how can the joint torques which minimize a particular cost functional be generated?
2. Generation of geometric paths: How does one generate a geometric path which avoids collisions with obstacles but can also be traversed at low cost?
3. Handling uncertainties: How sensitive will the generated torques be to variations in the robot's dynamic parameters, and how can these variations be taken into account at the trajectory planning stage?
4. Automatic calculation: Can all calculations be handled mechanically, and if so, is there a systematic way of generating a path planner for a given robot?

Solutions to these problems are the subjects of the next four chapters.

## CHAPTER 4

### TRAJECTORY PLANNING

The trajectory planning problem is basically an optimal control problem. One possible approach to the solution of this problem is to apply one of the standard tools of optimal control theory, Pontryagin's minimum principle. However, this approach requires solving a two-point boundary value problem for a non-linear system of differential equations with non-linear constraints; clearly, this does not lead to a tractable solution. The minimum principle also sheds little or no light on the other auxiliary problems, such as sensitivity to parameter variations. Therefore, we will take a more intuitive but systematic approach.

Three trajectory planners will be presented here. The first method will be referred to as the *phase plane method*. It is so called because it makes use of plots of the "pseudo-velocity"  $\mu \equiv \dot{\lambda}$  vs. the position parameter  $\lambda$ . (Recall that the robot is to move along a geometric path in which the joint positions  $q^i$  are given by a set of parametric functions, i.e.,  $q^i = f^i(\lambda)$ .) Such a plot, in which a velocity is plotted as a function of position, is generally referred to as a "phase plane plot", hence the name. Actually all three trajectory planners described here make use of this idea in one way or another, and from here on, the term "trajectory" will be taken to mean "phase trajectory", or  $\lambda$ - $\mu$  plot.

The phase plane method is in general applicable only to minimum time problems, but this is often the case in which we are most interested. Since only minimum-time solutions are to be considered, it is useful to consider how this restriction on the objective function can be used. Obviously, minimizing traversal times is equivalent to maximizing traversal speed. Given this fact, it is easy to see that, at least in the simplest case, the minimum time solution consists of an accelerating and a decelerating part; the robot should accelerate at its maximum rate, then "put on the brakes" at precisely that time which will bring it to a stop at the destination point. Of course, there will in general be some velocity limits as well as acceleration limits. The velocity limits are imposed by the interaction of velocity-dependent force terms in the dynamic equations and the actuator torque limits; the actuators must generate enough torque to overcome these forces and keep the manipulator on the desired path. If the robot is to avoid these velocity limits, then the trajectory must alternately accelerate and decelerate, and the switching points should be timed so that the trajectory just barely misses exceeding the velocity limits. A more precise description of this method appears in the next section of this chapter, including a derivation of the velocity limits and an algorithm for generating the optimal trajectories. Other complications are also discussed.

As an alternative, dynamic programming can be used to solve the trajectory planning problem. Dynamic programming is an impractical method for solving the general path planning problem for an arm with a large number of joints, since there are two state variables per joint, thus requiring a  $2n$  dimensional grid. (This is a classic example of the "curse of dimensionality".) However, when the path is given, there are only two state variables; thus only a 2-dimensional grid is required.

To use dynamic programming, the grid is set up so that the position parameter  $\lambda$  is used as the stage variable. Thus a "column" of the grid corresponds to a fixed value of  $\lambda$ , while a "row" corresponds to a fixed  $\mu$  value. One starts at the desired final state (the last column of the grid, with the row corresponding to the desired final  $\mu$  value) and assigns that state zero cost. All other states with position  $\lambda_{\max}$  are given a cost of infinity. Then the usual dynamic programming algorithm can be applied. The algorithm starts at the last column. For each point in the previous column one finds all the accessible points in the current column, determines the minimum cost to go from the previous to the current column, and increments costs accordingly. For each of the previous grid points, the optimal choice of the next grid point is recorded. When the initial state is reached, the optimal trajectory is found by following the pointer chain which starts at the given initial state. In the case at hand, determining which points are accessible from one column to the next is simply a matter of checking to see if the slope of the curve connecting the two points gives a permissible value. (The slope limits can be found from the limits on the actuator torques.) The incremental cost is easily computed for minimum time-energy problems, so a running sum can easily be kept for the total cost.

Dynamic programming has the advantage that it is a well-established and well-understood optimization method. It also gives the control law for *any* point on the curve, and so makes provision for the robot to vary its speed if necessary. (This of course assumes that the robot stays on the desired path.) On the other hand, if it is implemented in the most obvious and straightforward manner, it requires a large array for computations, and if the array size is to be known in advance then an upper bound on the velocity is needed. In practice, there may be artificially imposed



velocity bounds, but in general it would be necessary to either calculate velocity bounds in advance or create a new (larger) grid and start all over if the trajectory left the grid. The computation times also increase rather quickly as the density of the grid, and hence the accuracy of the solution, increases. Some modifications to the algorithm will be suggested which should considerably increase its speed.

It should also be noted that dynamic programming may still be used even if the robot's actuator torque constraints are not independent of one another; this is not the case with the phase plane method. Making the phase plane algorithm work for non-independent actuators would require that the space of actuator torques be searched for an acceleration bound. Dynamic programming only requires that a function be available which returns a yes-or-no answer to the question "if this acceleration is desired, will the required torques be realizable?". The dynamic programming algorithm itself performs the search of the actuator torque space.

A third algorithm, called the *perturbation trajectory improvement algorithm*, will be presented. This algorithm is in some respects similar to the dynamic programming algorithm, though like the phase plane method it is only applicable to minimum time problems, at least in the form in which it is presented here. This algorithm starts with an initial feasible trajectory, and perturbs the trajectory in such a way that the traversal time for the trajectory decreases, while the trajectory remains feasible. This method has most of the advantages of the dynamic programming method, and can be modified to generate minimum-time trajectories when there are limits on the *jerk*, or the derivative of the acceleration, as well as limits on joint torques.

#### 4.1. Parameterization of the Robot Dynamic Equations

Before delving headlong into the trajectory planning problem, the effects of restricting the manipulator's motion to a fixed path will be investigated. In what follows, the manipulator will be restricted to some geometric path

$$\mathbf{q}^i = f^i(\lambda), \quad 0 \leq \lambda \leq \lambda_{\max} \quad (4.1.1)$$

Plugging this into the dynamic equations (2.4.11) gives an expression for the velocity  $\mathbf{v}^i$ ,

$$\mathbf{v}^i = \dot{\mathbf{q}}^i = \frac{df^i}{d\lambda} \frac{d\lambda}{dt} = \frac{df^i}{d\lambda} \dot{\lambda} = \frac{df^i}{d\lambda} \mu \quad (4.1.2)$$

where  $\mu \equiv \dot{\lambda}$  is the *pseudo-velocity* of the manipulator. Equation (2.4.12), the equation for the joint torque/force vector, becomes

$$\begin{aligned} \mathbf{u}_i = & \mathbf{J}_{ij}(\lambda) \frac{df^j}{d\lambda} \dot{\mu} + \mathbf{J}_{ij}(\lambda) \frac{d^2 f^j}{d\lambda^2} \mu^2 \\ & + \mathbf{C}_{ijk}(\lambda) \frac{df^j}{d\lambda} \frac{df^k}{d\lambda} \mu^2 + \mathbf{R}_{ij} \frac{df^j}{d\lambda} \mu + \mathbf{g}_i(\lambda) \end{aligned} \quad (4.1.3)$$

The equations of motion along the curve (i.e., the geometric path) then become

$$\dot{\lambda} = \mu \quad (4.1.4)$$

$$\mathbf{u}_i = \mathbf{J}_{ij}(\lambda) \frac{df^j}{d\lambda} \dot{\mu} + \mathbf{J}_{ij}(\lambda) \frac{d^2 f^j}{d\lambda^2} \mu^2 \quad (4.1.5)$$

$$+ \mathbf{C}_{ijk}(\lambda) \frac{df^j}{d\lambda} \frac{df^k}{d\lambda} \mu^2 + \mathbf{R}_{ij} \frac{df^j}{d\lambda} \mu + \mathbf{g}_i(\lambda).$$

It is of course assumed that the coordinates  $q^i$  vary continuously with  $\lambda$ . It is also assumed that the derivatives  $\frac{df^i}{d\lambda}$  and  $\frac{d^2f^i}{d\lambda^2}$  exist, and that the derivatives  $\frac{df^i}{d\lambda}$  are never all zero simultaneously. This ensures that the path never retraces itself as  $\lambda$  goes from 0 to  $\lambda_{\max}$ . Such a retrace would force the parameter  $\lambda$  to take a discontinuous jump in order for the point  $q^i$  to move forward continuously.

It should be noted that in practice the spatial paths are given in Cartesian coordinates. While it is in general difficult to convert a curve in Cartesian coordinates to that in joint coordinates, it is relatively easy to perform the conversion for individual points. One can then pick a sufficiently large number of points on the Cartesian path, convert to joint coordinates, and use some sort of interpolation technique (e.g. cubic splines) to obtain a similar path in joint space (see [19] for an example).

Introducing some shorthand notation, let

$$M_i \equiv J_{ij} \frac{df^j}{d\lambda},$$

$$Q_i \equiv J_{ij} \frac{d^2f^j}{d\lambda^2} + C_{ijk} \frac{df^j}{d\lambda} \frac{df^k}{d\lambda},$$

$$R_i \equiv R_{ij} \frac{df^j}{d\lambda},$$

$$S_i \equiv g_i.$$

We then have

$$u_i = M_i \dot{\mu} + Q_i \mu^2 + R_i \mu + S_i \quad (4.1.6)$$

Note that the quantities listed above are functions of  $\lambda$ . For the sake of brevity, the

functional dependence is not indicated in what follows.

## 4.2. The Phase Plane Method

With the details of curve parameterization out of the way, the phase plane trajectory planning method may now be derived. (This derivation is substantially the same as that found in [30].) As was mentioned earlier in this chapter, the phase plane algorithm determines a series of alternately accelerating and decelerating phase trajectory segments. The acceleration and deceleration along these segments are the maximum allowable and minimum allowable values of the pseudo-acceleration  $\dot{\mu}$ . These values will now be derived.

### 4.2.1. Derivation of Pseudo-Acceleration Limits

Consider the constraints on the inputs, namely  $u_i^{\min}(\mathbf{q}, \dot{\mathbf{q}}) \leq u_i \leq u_i^{\max}(\mathbf{q}, \dot{\mathbf{q}})$ . The torque constraints along the parameterized curve can be found by inserting  $f^i(\lambda)$  for  $q^i$  and  $\frac{df^i}{d\lambda} \mu$  for  $\dot{q}^i$ . This gives constraints of the form  $u_i^{\min}(\lambda, \mu) \leq u_i \leq u_i^{\max}(\lambda, \mu)$ . The dynamic equations (4.1.6) can be viewed as having the form

$$u_i = \Psi_i(\lambda)\dot{\mu} + \Omega_i(\lambda, \mu)$$

where  $\Psi_i(\lambda) = M_i(\lambda)$  and  $\Omega_i(\lambda, \mu) = Q_i(\lambda)\mu^2 + R_i(\lambda)\mu + S_i(\lambda)$ . For a given state, i.e., given  $\lambda$  and  $\mu$ , this is just a set of parametric equations for a line, where the parameter is  $\dot{\mu}$ . The admissible controls, then, are those which are on this line in the input space and also are inside the rectangular prism formed by the input magnitude constraints. Thus the rectangular prism puts bounds on  $\dot{\mu}$ . The reason for

converting from bounds on the input torques/forces to bounds on the pseudo-acceleration  $\dot{\mu}$  is that all the positions, velocities, and accelerations of the various joints are related to one another through the parameterization of the path. Given the current state  $(\lambda, \mu)$ , the quantity  $\dot{\mu}$ , if known, determines the input torques/forces for *all* of the joints of the robot, so that manipulation of this one scalar quantity can replace the manipulation of  $n$  scalars (the input torques) and a set of constraints (the path parameterization equations).

For evaluating the bounds on  $\dot{\mu}$  explicitly, (4.1.6) can be plugged into the inequalities  $u_i^{\min} \leq u_i \leq u_i^{\max}$  so that

$$u_i^{\min}(\lambda, \mu) \leq M_i \dot{\mu} + Q_i \mu^2 + R_i \mu + S_i \leq u_i^{\max}(\lambda, \mu) \quad (4.2.1.1)$$

If  $M_i = 0$ , then these inequalities put no constraints on  $\dot{\mu}$ . However, the inertia matrix  $J_{ij}$  is positive definite, and by hypothesis the derivatives  $\frac{df^i}{d\lambda}$  are not all zero simultaneously at any point on the curve. Therefore,

$$J_{ij} \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} = M_i \frac{df^i}{d\lambda} > 0$$

for all values of  $\lambda$ . But then we must have at least one non-zero  $M_i$ , so that there will always be *some* constraint on the pseudo-acceleration. In those cases where  $M_i \neq 0$ , manipulation of inequalities (4.2.1.1) gives

$$\frac{u_i^{\min} - Q_i \mu^2 - R_i \mu - S_i}{|M_i|} \leq \text{sgn}(M_i) \dot{\mu} \leq \frac{u_i^{\max} - Q_i \mu^2 - R_i \mu - S_i}{|M_i|} \quad (4.2.1.2)$$

or

$$LB_i \leq \dot{\mu} \leq UB_i \quad (4.2.1.3)$$

where

$$LB_i \equiv \frac{u_i^{\min}(M_i > 0) + u_i^{\max}(M_i < 0) - (Q_i \mu^2 + R_i \mu + S_i)}{M_i} \quad (4.2.1.4)$$

and

$$UB_i \equiv \frac{u_i^{\max}(M_i > 0) + u_i^{\min}(M_i < 0) - (Q_i \mu^2 + R_i \mu + S_i)}{M_i} \quad (4.2.1.4)$$

The expression  $(M_i > 0)$  evaluates to one if  $M_i > 0$ , zero otherwise. Since these constraints must hold for all  $n$  joints,  $\dot{\mu}$  must satisfy

$$\max_i LB_i \leq \dot{\mu} \leq \min_i UB_i \quad \text{or} \quad GLB(\lambda, \mu) \leq \dot{\mu} \leq LUB(\lambda, \mu) \quad (4.2.1.6)$$

Note that it has *not* been assumed here that  $u_i^{\min}$  and  $u_i^{\max}$  are constants; they may indeed be arbitrary functions of  $\lambda$  and  $\mu$ . Later these quantities will be assumed to have specific, relatively simple forms, but these forms should be adequate to describe most of the actuators used in practice.

The difference between the trajectory planning algorithm to be presented and those which are conventionally used can be seen in terms of equation (4.2.1.6). Assume that the parameter  $\lambda$  is arc length in Cartesian space. Then  $\mu$  is the speed and  $\dot{\mu}$  the acceleration along the geometric path. Since most conventional trajectory planners put *constant* bounds on the acceleration over some set of position intervals, one would have  $GLB(\lambda, \mu) \leq \dot{\mu}_{\min} \leq \dot{\mu} \leq \dot{\mu}_{\max} \leq LUB(\lambda, \mu)$ , where  $\mu_{\min}$  and  $\mu_{\max}$  are constants. The conventional techniques, then, restrict the acceleration more than is really necessary. Likewise, constant bounds on the velocity will also be more

restrictive than necessary.

#### 4.2.2. Formulation of the Optimal Control Problem

With the manipulator dynamic equations and joint torque/force constraints in suitable form, we can address the actual control problem. Problems which require the minimization of cost functions subject to differential equation constraints can be expressed very naturally in the language of optimal control theory. The usual method of solving such a problem is to employ Pontryagin's maximum principle[6]. The maximum principle yields a two-point boundary value problem which is, except in some simple cases, impossible to solve in closed form, and usually is difficult to solve numerically as well. We will therefore not use the maximum principle, but will use some simpler reasoning, taking advantage of the specific form of the cost function and of the controlled system.

In the case considered here, minimum cost is equated with minimum time, thus maximizing the operating speed of the robot. The cost function can then be expressed as  $T = \int_0^{t_f} 1 \cdot dt$  where the final time  $t_f$  is left free. It is assumed here that the desired geometric path of the manipulator has been pre-planned, and is provided to the minimum-time controller in parametric form, as described earlier.

With this parameterization, there are two state variables, i.e.,  $\lambda$  and  $\mu$ , but  $(n+1)$  equations. One way to look at the system is to choose the equation  $\dot{\lambda} = \mu$  and one of the remaining equations as state equations, regarding the other equations as constraints on the inputs and on  $\dot{\mu}$ . However, the problem has a more appealing symmetry if a single differential equation is obtained from the  $n$  equations (4.1.6) by multiplying the  $i^{\text{th}}$  equation by  $\frac{df^i}{d\lambda}$  and sum over  $i$ , giving

$$\mathbf{u}_i \frac{df^i}{d\lambda} = M_i \frac{df^i}{d\lambda} \dot{\mu} + Q_i \frac{df^i}{d\lambda} \mu^2 + R_i \frac{df^i}{d\lambda} \mu + S_i \frac{df^i}{d\lambda} \quad (4.2.2.1)$$

or

$$U = M(\lambda)\dot{\mu} + Q(\lambda)\mu^2 + R(\lambda)\mu + S(\lambda) \quad (4.2.2.2)$$

where, expanding the values of  $M_i$ ,  $Q_i$ ,  $R_i$ , and  $S_i$ , we have

$$M(\lambda) \equiv \mathbf{J}_{ij}(\lambda) \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} \quad (4.2.2.3)$$

$$Q(\lambda) \equiv \mathbf{J}_{ij}(\lambda) \frac{df^i}{d\lambda} \frac{d^2 f^j}{d\lambda^2} + \mathbf{C}_{ijk}(\lambda) \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} \frac{df^k}{d\lambda} \quad (4.2.2.4)$$

$$R(\lambda) \equiv \mathbf{R}_{ij} \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} \quad (4.2.2.5)$$

$$S(\lambda) \equiv \mathbf{g}_i(\lambda) \frac{df^i}{d\lambda} \quad (4.2.2.6)$$

$$U(\lambda) \equiv \mathbf{u}_i \frac{df^i}{d\lambda} . \quad (4.2.2.7)$$

This formulation has a distinct advantage. Note that the coefficient of  $\dot{\mu}$  is quadratic in the vector of derivatives of the constraint functions. Since a smooth curve can always be parameterized in such a way that the first derivatives never all disappear simultaneously, and since the inertia matrix is positive definite, the whole equation can be divided by the non-zero, positive coefficient of  $\dot{\mu}$ , providing a solution for  $\dot{\mu}$  in terms of  $\lambda$  and  $\mu$ . Now there are only two state equations, and the original  $n$  dynamic equations can be regarded as constraints on the inputs and on  $\dot{\mu}$ .

The  $M$  term in (4.2.2.2) is a quadratic form reminiscent of the expression for the manipulator's kinetic energy. In fact, if the parametric expressions for the  $\dot{\mathbf{q}}_i$  are



plugged into the formula for kinetic energy, one obtains the expression  $K = \frac{1}{2} M \mu^2$ .

The  $Q$  term represents the components of the Coriolis and centrifugal forces which act along the path plus the fictitious forces generated by the restriction that the robot stay on the parameterized path. The  $R$  term represents frictional components, and  $S$  gives the gravitational force along the path.  $U$  is the projection of the input torque vector onto the velocity vector.

With this formulation, the state equations become

$$\dot{\lambda} = \mu \quad (4.2.2.8)$$

$$\dot{\mu} = \frac{1}{M} [U - Q \mu^2 - R \mu - S] \quad (4.2.2.9)$$

The traversal time of the path,  $T$ , can be written in terms of  $\lambda$  and  $\mu$  as

$$T = t_f = \int_0^{t_f} 1 \cdot dt = \int_0^{\lambda_{\max}} \frac{dt}{d\lambda} d\lambda = \int_0^{\lambda_{\max}} \frac{1}{\mu(\lambda)} d\lambda \quad (4.2.2.10)$$

Given these forms for the dynamic equations and the cost function, we have the Minimum Time Path Planning (MTPP) problem as follows.

**Problem MTPP:**

Find  $\mu^*(\lambda)$  and  $u_i^*(\lambda, \mu^*(\lambda))$  by minimizing  $T$  subject to (4.2.2.8), (4.2.2.9),  $u_i^{\min}(\lambda, \mu^*(\lambda)) \leq u_i^*(\lambda, \mu^*(\lambda)) \leq u_i^{\max}(\lambda, \mu^*(\lambda))$ ,  $0 \leq \lambda \leq \lambda_{\max}$ , and the boundary conditions  $\mu^*(0) = \mu_0$  and  $\mu^*(\lambda_{\max}) = \mu_f$ .

### 4.2.3. Phase Plane Plots

At this point, it is instructive to look at the system's behavior in the phase plane. The equations of the phase plane trajectories can be obtained by dividing equation (4.2.2.9) by (4.2.2.8). This gives

$$\frac{d\mu}{d\lambda} = \frac{\frac{d\mu}{dt}}{\frac{d\lambda}{dt}} = \frac{\dot{\mu}}{\mu} = \frac{1}{\mu M} [U - Q\mu^2 - R\mu - S] \quad (4.2.3.1)$$

Noting again that the total time  $T$  that it takes to go from initial to final states is

$T = \int_0^{\lambda_{\max}} \frac{1}{\mu} d\lambda$ , it is easily seen that minimizing time requires that the pseudo-velocity  $\mu$  be made as large as possible, a result which would be expected intuitively.

The constraints on  $\dot{\mu}$  have two effects. One effect is to place limits on the slope of the phase trajectory. The other is to place limits on the value of  $\mu$ . To obtain the limits on  $\frac{d\mu}{d\lambda}$ , one simply divides the limits on  $\dot{\mu}$  by  $\mu$ , since  $\frac{d\mu}{d\lambda} = \frac{\dot{\mu}}{\mu}$ .

To get the constraints on  $\mu$ , it is necessary to consider the bounds on  $\mu$ . If, for particular values of  $\lambda$  and  $\mu$ , we have  $LUB(\lambda, \mu) < GLB(\lambda, \mu)$  then there are no permissible values of  $\dot{\mu}$ . Therefore, for each value of  $\lambda$  we can assign a set of values of  $\mu$  as determined by the inequality  $LUB(\lambda, \mu) - GLB(\lambda, \mu) \geq 0$ . This inequality holds if and only if  $UB_i(\lambda, \mu) - LB_j(\lambda, \mu) \geq 0$  for all  $i$  and  $j$ . The intersection of the regions determined by these inequalities produces a region of the phase plane outside of which the phase trajectory must not stray. This region will hereafter be referred to as the *admissible region* of the phase plane. Using the equations for the lower and upper bounds for all  $i$  and  $j$ ,

$$\frac{u_i^{\max}(M_i > 0) + u_i^{\min}(M_i < 0) - (Q_i \mu^2 + R_i \mu + S_i)}{M_i} \quad (4.2.3.2)$$

$$- \frac{u_j^{\min}(M_j > 0) + u_j^{\max}(M_j < 0) - (Q_j \mu^2 + R_j \mu + S_j)}{M_j} \geq 0$$

Rearranging this inequality,

$$\left[ \frac{Q_i}{M_i} - \frac{Q_j}{M_j} \right] \mu^2 + \left[ \frac{R_i}{M_i} - \frac{R_j}{M_j} \right] \mu + \left[ \frac{S_i}{M_i} - \frac{S_j}{M_j} \right] \quad (4.2.3.3)$$

$$+ \left[ \frac{u_i^{\max}(M_i < 0) - u_i^{\min}(M_i > 0)}{|M_i|} - \frac{u_j^{\min}(M_j < 0) - u_j^{\max}(M_j > 0)}{|M_j|} \right] \geq 0$$

It will prove convenient to “symmetrize” the input torque bounds in the discussion which follows. Each joint has a mean torque  $u_M^i$  and a maximum deviation  $\Delta^i$  given by  $u_M^i \equiv \frac{u_i^{\max} + u_i^{\min}}{2}$ ,  $\Delta^i \equiv \frac{u_i^{\max} - u_i^{\min}}{2}$ . The inequality (4.2.3.3) can

then be rewritten as

$$\left[ \frac{Q_i}{M_i} - \frac{Q_j}{M_j} \right] \mu^2 + \left[ \frac{R_i}{M_i} - \frac{R_j}{M_j} \right] \mu \quad (4.2.3.4)$$

$$+ \left[ \frac{S_i}{M_i} - \frac{S_j}{M_j} \right] - \left[ \frac{u_M^i}{M_i} - \frac{u_M^j}{M_j} \right] + \left[ \frac{\Delta^i}{|M_i|} + \frac{\Delta^j}{|M_j|} \right] \geq 0$$

At this point, a specific form for the torque bounds will be assumed. If the maximum and minimum torques for each joint are functions only of the states  $q^i$  and  $\dot{q}^i$  (i.e., the actuator torques are all independent of one another) and are at most *quadratic* in the velocities  $\dot{q}^i$ , then this inequality yields a simple quadratic in  $\mu$ . This allows one to solve for the velocity bounds using the quadratic formula. A particularly simple and useful special case is that encountered when the actuator is a fixed-field D.C. motor with a bounded voltage input. In this case, the torque constraints take the form  $u_i^{\max} = V_{\max}^i + k_m^i \dot{q}^i$  and  $u_i^{\min} = V_{\min}^i + k_m^i \dot{q}^i$  where  $V_{\min}^i$  and  $V_{\max}^i$  are proportional to the voltage limits and  $k_m^i$  is a constant which depends upon the motor winding resistance, voltage source resistance, and the back E.M.F. generated by the motor. Let  $V_{ave} = \frac{V_{\max}^i + V_{\min}^i}{2}$  and  $\Delta^i = \frac{V_{\max}^i - V_{\min}^i}{2}$ . Then,

we get

$$u_M^i = V_{ave} + k_m^i \dot{q}^i = V_{ave} + k_m^i \frac{df^i}{d\lambda} \mu \quad (4.2.3.5)$$

From here on, the case outlined above will be used for the sake of simplicity. The only changes required for the more general case of quadratic velocity dependence of the torque bounds is a re-definition of the coefficients in some of the equations which follow.

Introducing yet more shorthand notation, let

$$A_{ij} \equiv \left[ \frac{Q_i}{M_i} - \frac{Q_j}{M_j} \right] \quad (4.2.3.6)$$

$$B_{ij} \equiv \left[ \frac{R_i}{M_i} - \frac{R_j}{M_j} \right] - \left[ \frac{V_{ave} + k_m^i \frac{df^i}{d\lambda} \mu}{M_i} - \frac{V_{ave} + k_m^j \frac{df^j}{d\lambda} \mu}{M_j} \right] \quad (4.2.3.7)$$

$$C_{ij} \equiv \left[ \frac{\Delta^i}{|M_i|} + \frac{\Delta^j}{|M_j|} \right] \quad (4.2.3.8)$$

$$D_{ij} \equiv \left[ \frac{S_j}{M_j} - \frac{S_i}{M_i} \right] \quad (4.2.3.9)$$

Noting that (at least in this case)  $A_{ij} = -A_{ji}$ ,  $B_{ij} = -B_{ji}$ ,  $C_{ij} = C_{ji}$ , and  $D_{ij} = -D_{ji}$ , we have the inequalities

$$A_{ij} \mu^2 + B_{ij} \mu + C_{ij} + D_{ij} \geq 0 \quad (4.2.3.10)$$

$$-A_{ij} \mu^2 - B_{ij} \mu + C_{ij} - D_{ij} \geq 0 \quad (4.2.3.11)$$

The second inequality is obtained by interchanging  $i$  and  $j$  and using the symmetry or anti-symmetry of the coefficients. Only the cases where  $i \neq j$  need be considered, so there are  $\frac{n(n-1)}{2}$  such pairs of equations, where  $n$  is the number of degrees of freedom of the robot.

If  $A_{ij} = B_{ij} = 0$ , we have  $C_{ij} - D_{ij} \geq 0$  and  $C_{ij} + D_{ij} \geq 0$ , which are always true if the robot is "strong" enough so that it can stop and hold its position at all points on the desired path. If  $A_{ij} = 0$  and  $B_{ij} \neq 0$ , then we have a pair of linear inequalities which determine a closed interval for  $\mu$ . If  $A_{ij} \neq 0$ , then, without loss of generality, we can assume that  $A_{ij} > 0$ . Then the left-hand side of the first of the inequalities (4.2.3.10) is a parabola which is concave upward, whereas for the

second it is concave downward. When the parabola is concave downward, then the inequality holds when  $\mu$  is between the two roots of the quadratic. If the parabola is concave upward, then the inequality holds outside of the region between the roots (Figure 4.2.1). Thus in one case  $\mu$  must lie within a closed interval and in the other it must lie outside an open interval, unless of course the open interval is of length zero. In that case, the inequality constraint is always satisfied and the roots of the quadratic will be complex.

Since the admissible values of  $\mu$  are those which satisfy all of the inequalities, the admissible values must lie in the intersection of all the regions determined by the inequalities. There are  $\frac{n(n-1)}{2}$  inequalities which give closed intervals, so the intersection of these regions is also a closed interval. The other  $\frac{n(n-1)}{2}$  inequalities, when intersected with this closed interval, each may have the effect of "punching a hole" in the interval (Figure 4.2.2). It is thus possible to have, for any particular value of  $\lambda$ , a set of admissible values for  $\mu$  which consists of as many as  $\frac{n(n-1)+2}{2}$  distinct intervals. When the phase portrait of the optimal path is drawn, it may be necessary to have the optimal trajectory dodge the little "islands" which can occur in the admissible region of the phase plane. (Hereafter, these inadmissible regions will be referred to as *islands of inadmissibility*, or just *islands*.) It should be noted, though, that if there is no friction, then  $B_{jj} = 0$ , which means that in the concave upward case the inequality is satisfied for all values of  $\mu$ . Thus in this case there will be no islands in the admissible region.

In addition to the constraints on  $\mu$  described above, we must also have  $\mu \geq 0$ . This can be shown as follows: if  $\mu < 0$ , then the trajectory has passed below the line

$\mu = 0$ . Below this line, the trajectories always move to the left, since  $\mu \equiv \frac{d\lambda}{dt} < 0$ .

Since the optimal trajectory must approach the desired final state through positive values of  $\mu$ , the trajectory would then have to pass through  $\mu = 0$  again, and would pass from  $\mu < 0$  to  $\mu > 0$  at a point to the left of where it had passed from  $\mu > 0$  to  $\mu < 0$ . Thus in order to get to the desired final state, the trajectory would have to cross itself, forming a loop. But, then, there is no sense in traversing the loop; it would take less time to just use the crossing point as a switching point. Thus we need consider only those points of the phase plane for which  $\mu \geq 0$ .

Another way of thinking about the system phase portrait is to assign a pair of vectors to each point in the phase plane. One vector represents the slope when the system is accelerating (i.e.,  $\dot{\mu}$  is maximized) and the other represents the slope for deceleration (i.e.,  $\dot{\mu}$  is minimized). This pair of vectors looks like a pair of scissors, and as the position in the phase plane changes, the angles of both the upper and lower jaws of the pair of scissors change. In particular, the angle between the two vectors varies with position. The phase trajectories must, at every point of the phase plane, point in a direction which lies between the jaws of the scissors. At particular points of the phase plane, though, the jaws of the scissors close completely, allowing only a single value for the slope. At other points the scissors may try to go past the closed position, allowing no trajectory at all. This phenomenon, and the condition  $\mu \geq 0$ , determine the admissible region of the phase plane. This is illustrated in Figure 4.2.3. Note that the boundary of the admissible region passes through those points which have only a single vector associated with them, corresponding to those states where only a single acceleration value is permitted.

#### 4.2.4. Determination of Optimal Trajectories

For illustrative purposes, we first present an algorithm for finding the optimal trajectories for which there are no islands in the phase plane which need to be dodged. The only restrictions, then, will be that  $\mu$  must lie between a pair of values which are easily calculable, given  $\lambda$ . The optimal trajectory can be constructed by the following steps called the *Algorithm for Constructing Optimal Trajectories, No Islands (ACOTNI)*.

- S1. Start at  $\lambda=0$ ,  $\mu=\mu_0$  and construct a trajectory that has the maximum acceleration value. Continue this curve until it either leaves the admissible region of the phase plane or goes past  $\lambda=\lambda_{\max}$ . Note that "leaves the admissible region" implies that if part of the trajectory happens to coincide with a section of the admissible region's boundary, then the trajectory should be extended along the boundary. It is not sufficient in this case to continue the trajectory only until it touches the edge of the admissible region.
- S2. Construct a second trajectory that starts at  $\lambda=\lambda_{\max}$ ,  $\mu=\mu_f$  and proceeds *backwards*, so that it is a decelerating curve. This curve should be extended until it either leaves the admissible region or extends past  $\lambda=0$ .
- S3. If the two trajectories intersect, then stop. The point at which the trajectories intersect is the (single) switching point, and the optimal trajectory consists of the first (accelerating) curve from  $\lambda=0$  to the switching point, and the second (decelerating) curve from the switching point to  $\lambda=\lambda_{\max}$  (Figure 4.2.4).
- S4. If the two curves under consideration do not intersect, then they must both leave the admissible region. Call the point where the accelerating curve leaves



the admissible region  $\lambda_1$ . This is a point on the boundary curve of the admissible region (Figure 4.2.5). If the boundary curve is given by  $\mu = g(\lambda)$ , then search along the curve, starting at  $\lambda_1$ , until a point is found at which the quantity  $\phi(\lambda) \equiv \frac{d\mu}{d\lambda} - \frac{dg}{d\lambda}$  changes sign. (Note that since  $g(\lambda)$  determines the boundary of the admissible region, there is only one allowable value of  $\frac{d\mu}{d\lambda}$ . Also note that if  $g(\lambda)$  has a discontinuity,  $\frac{dg}{d\lambda}$  must be treated as  $+\infty$  or  $-\infty$  depending upon the direction of the jump.) This point is the next switching point. Call it  $\lambda_d$ .

- S5. Construct a decelerating trajectory backwards from  $\lambda_d$  until it intersects an accelerating trajectory. This gives another switching point (see point A in Figure 4.2.6).
- S6. Construct an accelerating trajectory starting from  $\lambda_d$ . Continue the trajectory until it either intersects the final decelerating trajectory or it leaves the admissible region. If it intersects the decelerating trajectory, then the intersection gives another switching point (see point C in Figure 4.2.6), and the procedure terminates. If the trajectory leaves the admissible region, then go to S4.

This algorithm yields a sequence of alternately accelerating and decelerating curves which give the optimal trajectory. Before discussing the optimality of the trajectory, one has to show that all steps of the ACOTNI are possible and that the ACOTNI will terminate.

Addressing the first question, S1, S2, S3, S5, and S6 are clearly possible. S4 requires finding a sign change of the function  $\phi(\lambda)$ . Since  $\phi(\lambda)$  must be greater than zero where the accelerating trajectory leaves the admissible region and less than zero where the decelerating trajectory leaves, there must be a sign change. Therefore all

steps are possible.

In order to prove that ACOTNI terminates, it must be shown that the search for switching points in step S4 will be performed a finite number of times. To prove this, it is sufficient to prove that the number of isolated zeros of  $\phi(\lambda)$ , the number of intervals of positive extent over which  $\phi(\lambda)$  is zero, and the number of intervals over which  $\phi(\lambda)$  does not exist are all finite. To do this, some assumption about the form of the functions  $f^i(\lambda)$  must be made. It will be assumed here that the  $f^i$  are real valued, piecewise analytic, and composed of a finite number of pieces. (In other words, the  $f^i$  are analytic splines.) Under these assumptions, the following theorem proves the convergence of ACOTNI within a finite number of iterations.

**Theorem 4.2.1:** If the functions  $f^i$  are composed of a finite number of analytic, real-valued pieces, then the function  $\phi(\lambda)$  has a finite number of intervals over which it is identically zero and a finite number of zeros outside those intervals.

**Proof:** The inertia matrix, Coriolis array, and gravitational loading vector are all piecewise analytic in the  $q^i$ , and since the  $f^i(\lambda)$  are analytic in  $\lambda$ , the inertia matrix, etc. when expressed as functions of  $\lambda$  (as in Eqs. (4.2.2.3) and (4.2.2.4)) are piecewise analytic and have a finite number of analytic pieces. The functions  $M_i, Q_i, R_i, S_i$  of Eq. (4.1.6) are, therefore, also piecewise analytic. Since a real-valued analytic function with no singularities in a finite interval must either have a finite number of zeros in that interval or be identically zero, the quantities  $M_i$  must either be identically zero in the interval considered or have a finite number of zeros. We cannot have all of the  $M_i$  zero, for if that were the case we would have

$$J_{ij} \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} = M_i \frac{df^i}{d\lambda} = 0$$
, which is not allowed by hypothesis. If only one of

the  $M_i$  is non-zero, then there is no boundary curve to deal with, and so no zeros. With two or more not identically zero, there will be a boundary curve. The curve is given by one of the equations (4.2.3.10) (with " $\geq$ " replaced by " $=$ ") for some pair of indices  $i$  and  $j$ . Since the coefficients  $A, B, C$ , and  $D$  in Eq. (4.2.3.10) are analytic except at the zeros of the  $M_i$ , and because the  $M_i$  have a finite number of zeros, we can divide the interval under consideration further, using the zeros of the  $M_i$  as division points. Within each subinterval, then, only one of the equations (4.2.3.10) holds. Since Eq. (4.2.3.10) determines  $\mu$  as an analytic function of  $\lambda$  within this interval, the bounding curve  $g(\lambda)$  is piecewise analytic. The curve  $\phi(\lambda)$ , then, is also piecewise analytic and is either identically zero or has a finite number of zeros in each subinterval. Thus, since  $\phi(\lambda)$  either is identically zero in each subinterval or has a finite number of zeros in the subinterval, the number of subintervals is finite, and the number of intervals is finite, the number of zeros and zero-intervals is finite. ■

Finally, the following theorem proves the optimality of the solution generated by the ACOTNI.

**Theorem 4.2.2:** Any trajectory generated by the ACOTNI is optimal in the sense of minimum time control.

**Proof:** Proof of this theorem is straightforward. First, make three observations: (i) From the form (4.2.2.10) of the cost  $T$ , there must be some  $\lambda_0$  such that the point  $(\lambda_0, \mu')$  on the new trajectory is higher than the point  $(\lambda_0, \mu)$  on the ACOTNI trajectory, i.e.,  $\mu' > \mu$ . Otherwise, we would not have a trajectory with a smaller travel time. (ii) The trajectory produced by ACOTNI consists of alternately accelerating and decelerating segments, and can therefore be divided into sections which consist of one accelerating and one decelerating segment. (iii) The admissible

portions of these sections which lie above the ACOTNI trajectory are bounded on the left and right by either the line  $\lambda = 0$ , the line  $\lambda = \lambda_{\max}$ , the boundary of the admissible region, or the ACOTNI trajectory itself. Now consider the point  $(\lambda, \mu')$  and the trajectory on which it lies. This trajectory, if extended backward and forward from  $(\lambda_0, \mu')$  must intersect a single section of the ACOTNI trajectory at two or more points, since otherwise it would either leave the admissible region or not meet the initial or final boundary conditions. One such point must occur for  $\lambda < \lambda_0$  and one must occur for  $\lambda > \lambda_0$ . But since the accelerating segment of the trajectory precedes the decelerating segment, the new trajectory must either intersect the accelerating part of the ACOTNI trajectory twice, intersect the decelerating part twice, or first intersect the accelerating part then the decelerating part. But since the torques were chosen so as to minimize or maximize  $U$  in equation (4.2.3.1), any of these situations leads to a contradiction of a theorem on differential inequalities presented in [18], pp. 41-43. ■

The whole idea of the algorithm is to generate trajectories which come as close as possible to the edge of the admissible region without actually passing outside it. Thus the trajectories just barely touch the inadmissible region. In practice this would, of course, be highly dangerous, since minute errors in the control inputs or measured system parameters would very likely make the robot stray from the desired path. Theoretically, however, this trajectory is the minimum-time optimum.

We are now in a position to consider the general case, i.e., the case in which friction, copper losses in the drive motor, etc., are sufficient to cause islands in the phase plane. In this case, the algorithm is most easily presented in a slightly different form. Since there may be several boundary curves instead of one, it is not possible to

search a single function for zeros, as was done in ACOTNI. Thus instead of looking for zeros as the algorithm progresses, we look for them all at once instead, and then construct the trajectories which “just miss” the boundaries, whether the boundaries be the edges of the admissible region or the edges of islands. The appropriate trajectories can then be found by searching the resulting directed graph, always taking the highest trajectory possible, and backtracking when necessary. More formally, the *Algorithm for Construction of Optimal Trajectories (ACOT)* is:

- S1. Construct the initial accelerating trajectory. (Same as ACOTNI.)
- S2. Construct the final decelerating trajectory. (Same as ACOTNI.)
- S3. Calculate the function  $\phi(\lambda)$  for the edge of the admissible region and for the edges of all the islands. At each of the sign changes of  $\phi(\lambda)$ , construct a trajectory for which the sign change is a switching point, as in ACOTNI S5 and S6. The switching direction (acceleration-to-deceleration or vice-versa) should be chosen so that the trajectory does not leave the admissible region. Extend each trajectory until it either leaves the admissible region, or goes past  $\lambda_{\max}$ .
- S4. Find all intersections of the trajectories. These are potential switching points.
- S5. Starting at  $\lambda=0$ ,  $\mu=\mu_0$ , traverse the grid formed by the various trajectories in such a way that the highest trajectory from the initial to the final points is followed. This is described below in the *grid traversal algorithm (GTA)*.

Traversing the grid formed by the trajectories generated in S3 and S4 above is a search of a directed graph, where the goal to be searched for is the final decelerating trajectory. If one imagines searching the grid by walking along the trajectories, then one would try to keep making left turns, if possible. If a particular turn lead to

a dead end, then it would be necessary to backtrack, and take a right turn instead. The whole procedure can best be expressed recursively, in much the same manner as tree traversal procedures.

The algorithm consists of two procedures, one which searches accelerating curves and one which searches decelerating curves. The algorithm is:

**AccSearch:**

On the current (accelerating) trajectory, find the last switching point. At this point, the current trajectory meets a decelerating curve. If that curve is the final decelerating trajectory, then the switching point under consideration is a switching point of the final optimal trajectory. Otherwise, call **DecSearch**, starting at the current switching point. If **DecSearch** is successful, then the current point is a switching point of the optimal trajectory. Otherwise, move back along the current accelerating curve to the previous switching point and repeat the process.

**DecSearch:**

On the current (decelerating) trajectory, find the first switching point. Apply **AccSearch**, starting on this point. If successful, then the current point is a switching point of the optimal trajectory. Otherwise, move forward to the next switching point and repeat the process.

These two algorithms always look first for the curves with the highest velocity, since **AccSearch** always starts at the end of an accelerating curve and **DecSearch** always starts at the beginning of a decelerating curve. Therefore the algorithm finds (if possible) the trajectory with the highest velocity, and hence the smallest traversal

time.

The proofs of optimality and convergence of this algorithm are virtually identical to those of ACOTNI, and will not be repeated here. Note that in the convergence proof for ACOTNI the fact that there is only a single boundary curve in the zero-friction case is never used; the same proof therefore applies in the high-friction case.

#### 4.2.5. Numerical Examples

To show how the minimum-time algorithm works, a numerical example follows. The robot used in the example is a simple two-degree-of-freedom robot with one revolute and one prismatic joint, i.e., a robot which moves in polar coordinates. Despite its simplicity, the example robot is sufficient to show most important aspects of the phase plane trajectory planning method. (More complicated examples are given later.) The path chosen is a straight line. Before applying the minimum-time algorithm, we must derive the dynamic equations for the robot. This requires calculation of the inertia matrix, so masses and moments of inertia of the robot must be given.

A drawing of the hypothetical robot is shown in Figure 4.2.7. The robot consists of a rotating fixture with moment of inertia  $J_\theta$  through which slides a uniformly dense rod of length  $L_r$  and mass  $M_r$ . The payload has mass  $M_p$  and moment of inertia  $J_p$ , and its center of mass is at the point  $(x, y)$  which is  $L_p$  units of length from the end of the sliding rod. (The full dynamic equations for this arm are derived in the Appendix.)

In the examples presented here, the robot will be moved from the point (1,1) to the point (1,-1). The equation of the curve can be expressed as  $r = \sec \theta$ , where  $\theta$  ranges from  $+\frac{\pi}{4}$  to  $-\frac{\pi}{4}$ . Introducing the parameter  $\lambda$ , one possible parameterization is

$$\theta = \frac{\pi}{4} - \lambda \quad r = \sec\left(\frac{\pi}{4} - \lambda\right) \quad (4.2.5.1)$$

Now introduce the shorthand expressions  $M_t \equiv M_r + M_p$ ,  $K \equiv M_r(L_r + 2L_p)$ , and  $J_t \equiv J_\theta + J_p + M_r(L_p^2 + L_r L_p + \frac{L_r^2}{3})$ . Plugging these expressions and the expressions for the derivatives of  $r$  and  $\theta$  into the dynamic equations for the polar manipulator gives (see the Appendix for a detailed derivation)

$$\mathbf{u}_r = -M_t \sec\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right) \dot{\mu} - k_r \sec\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right) \mu \quad (4.2.5.2)$$

$$+ \left[ M_t \sec\left(\frac{\pi}{4} - \lambda\right) \left( \sec^2\left(\frac{\pi}{4} - \lambda\right) + \tan^2\left(\frac{\pi}{4} - \lambda\right) \right) + \frac{K}{2} - M_t \sec\left(\frac{\pi}{4} - \lambda\right) \right] \mu^2$$

$$+ \mu^2 \left[ 2M_t \sec\left(\frac{\pi}{4} - \lambda\right) - K \right] \sec\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right)$$

Solving for  $\dot{\mu}$ , we have



$$\mu = \frac{-1}{M_t \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda)} \left[ u_r + k_r \mu \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda) \right. \quad (4.2.5.4)$$

$$\left. - \mu^2 \left\{ M_t \sec(\frac{\pi}{4} - \lambda) (\sec^2(\frac{\pi}{4} - \lambda) + \tan^2(\frac{\pi}{4} - \lambda)) + \frac{K}{2} - M_t \sec(\frac{\pi}{4} - \lambda) \right\} \right]$$

and

$$\dot{\mu} = \frac{u_\theta + k_\theta \mu - \mu^2 \left\{ 2M_t \sec(\frac{\pi}{4} - \lambda) - K \right\} \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda)}{J_t - K \sec(\frac{\pi}{4} - \lambda) + M_t \sec^2(\frac{\pi}{4} - \lambda)} \quad (4.2.5.5)$$

The signs of the coefficients of  $u_r$  and  $u_\theta$  are

$$\text{sgn}(u_r) = \begin{cases} -1 & 0 < \lambda < \frac{\pi}{4} \\ +1 & \frac{\pi}{4} < \lambda < \frac{\pi}{2} \end{cases} \quad \text{and} \quad \text{sgn}(u_\theta) = 1 \quad (4.2.5.6)$$

The limits on  $\dot{\mu}$  imposed by the  $\theta$  joint are the same over the whole interval.

For simplicity, let  $u_{\max}^i = -u_{\max}^i$  for  $i = r, \theta$ , then the limits are

$$\dot{\mu} \leq \frac{u_{\max}^\theta + \left[ \{ 2M_t \sec(\frac{\pi}{4} - \lambda) - K \} \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda) \right] \mu^2 - k_\theta \mu}{J_t - K \sec(\frac{\pi}{4} - \lambda) + M_t \sec^2(\frac{\pi}{4} - \lambda)} \quad (4.2.5.7)$$

and

$$\dot{\mu} \geq \frac{-\mathbf{u}_{\max}^{\theta} + \left[ \{2M_t \sec(\frac{\pi}{4} - \lambda) - K\} \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda) \right] \mu^2 - k_{\theta} \mu}{J_t - K \sec(\frac{\pi}{4} - \lambda) + M_t \sec^2(\frac{\pi}{4} - \lambda)} \quad (4.2.5.8)$$

For the r joint, consider the case when  $\lambda < \frac{\pi}{4}$ . Then we also have

$$\begin{aligned} \dot{\mu} \leq & \frac{1}{M_t \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda)} \left\{ \mathbf{u}_{\max}^r \right. \\ & + \left[ 2M_t \sec(\frac{\pi}{4} - \lambda) \tan^2(\frac{\pi}{4} - \lambda) + \frac{K}{2} \right] \mu^2 \\ & \left. - k_r \mu \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda) \right\} \end{aligned} \quad (4.2.5.9)$$

and

$$\begin{aligned} \mu \geq & \frac{1}{M_t \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda)} \left\{ -\mathbf{u}_{\max}^r \right. \\ & + \left[ 2M_t \sec(\frac{\pi}{4} - \lambda) \tan^2(\frac{\pi}{4} - \lambda) + \frac{K}{2} \right] \mu^2 \\ & \left. - k_r \mu \sec(\frac{\pi}{4} - \lambda) \tan(\frac{\pi}{4} - \lambda) \right\} \end{aligned} \quad (4.2.5.10)$$

For  $\lambda > \frac{\pi}{4}$  the limits have the signs of  $u_{\max}^r$  reversed.

Equating upper and lower limits on  $\dot{\mu}$  gives the boundary of the admissible region. For  $\lambda < \frac{\pi}{4}$ , Eqs. (4.2.5.8) and (4.2.5.9) give

$$A \mu^2 + B \mu + C \geq 0 \quad (4.2.5.11)$$

where

$$A = -KM_t \sec^4\left(\frac{\pi}{4} - \lambda\right) + 2M_t \sec^3\left(\frac{\pi}{4} - \lambda\right) + \frac{3}{2} KM_t \sec^2\left(\frac{\pi}{4} - \lambda\right) \quad (4.2.5.12)$$

$$- (2M_t J_t + \frac{K^2}{2}) \sec\left(\frac{\pi}{4} - \lambda\right) + \frac{KJ_t}{2}$$

$$B = (J_t k_r - M_t k_\theta) \sec\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right) - Kk_r \sec^2\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right) \quad (4.2.5.13)$$

$$+ M_t k_r \tan\left(\frac{\pi}{4} - \lambda\right) \sec^3\left(\frac{\pi}{4} - \lambda\right)$$

$$C = u_{\max}^r (J_t - K \sec\left(\frac{\pi}{4} - \lambda\right) + M_t \sec^2\left(\frac{\pi}{4} - \lambda\right)) \quad (4.2.5.14)$$

$$+ u_{\max}^\theta M_t \tan\left(\frac{\pi}{4} - \lambda\right) \sec\left(\frac{\pi}{4} - \lambda\right)$$

Likewise, Eqs. (4.2.5.7) and (4.2.5.10) give

$$-A \mu^2 - B \mu + C \geq 0 \quad (4.2.5.15)$$

The same inequalities, with  $u_{\max}^r$  negated, work when  $\lambda \geq \frac{\pi}{4}$ .

Finally, we need to determine the differential equations to be solved. These equations are

$$\dot{\mu} = \frac{1}{J_t - K \sec\left(\frac{\pi}{4} - \lambda\right) + M_t \sec^4\left(\frac{\pi}{4} - \lambda\right)} \left[ -u_\theta \right. \\ \left. - u_r \sec\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right) + 2\mu^2 M_t \sec^4\left(\frac{\pi}{4} - \lambda\right) \tan\left(\frac{\pi}{4} - \lambda\right) \right. \\ \left. - \left\{ k_r \tan^2\left(\frac{\pi}{4} - \lambda\right) \sec^2\left(\frac{\pi}{4} - \lambda\right) + k_\theta \right\} \mu \right] \quad (4.2.5.16)$$

$$\dot{\lambda} = \mu \quad (4.2.5.17)$$

The numerical values of the various constants which describe the robot are given in the Appendix in Table A.1. Using these data, the differential equations were solved numerically using the fourth-order Runge-Kutta method, the program being written in C and run under the UNIX<sup>1</sup> operating system on a VAX-11/780<sup>2</sup>. The derivative of the boundary curve  $g(\lambda)$  (needed to compute the function  $\phi(\lambda)$ ) was calculated numerically, and the sign changes of  $\phi(\lambda)$  found by bisection. The graphs of the resulting trajectories and of the boundary of the admissible region are given in Figure 4.2.8 for the zero-friction case and in Figures 4.2.9 and 4.2.10 for the high-friction case.

---

<sup>1</sup>UNIX is a trademark of Bell Laboratories.

<sup>2</sup>VAX is a trademark of Digital Equipment Corporation.

Note in particular the shape of the admissible region boundary in Figure 4.2.9. For values of  $\lambda$  less than about 0.42 there is not a single range of admissible velocities, but two ranges. Thus there is an "island" in the phase plane, though the island is chopped off by the constraint that  $\lambda$  be positive. While the existence of such islands may at first seem to defy intuition, the example shows that they do indeed exist. In this case, the island does not really come into play in the calculation of the optimal trajectory. Nevertheless, the example does demonstrate that there may be situations where the admissible region has a fairly complicated shape. Since most practical manipulators have more than two joints and have more complicated dynamic equations than those of the simple robot used here, it is conceivable that the admissible region of the phase plane for a practical robot arm could have quite a complicated shape.

As a final example, to demonstrate clearly the existence of islands in the phase plane, we include a sketch of the admissible region of the phase plane for a two-dimensional Cartesian robot moving along a circular path. In this case, the dynamic equations are a simple pair of uncoupled, linear differential equations with constant coefficients, i.e.,  $u_x = m\ddot{x} + k_x \dot{x}$ ,  $u_y = m\ddot{y} + k_y \dot{y}$  where  $m \equiv$  mass of  $x$  and  $y$  joints,  $k_x \equiv$  coefficient of friction of  $x$  joint, and  $k_y \equiv$  coefficient of friction of  $y$  joint.

Moving this manipulator in a unit circle, say in the first quadrant, requires that

$$x = \cos\lambda, \quad y = \sin\lambda, \quad 0 \leq \lambda \leq \frac{\pi}{2} \quad (4.2.5.18)$$

Plugging these expressions and their derivatives into the dynamic equations gives

$$\mathbf{u}_x = -m \dot{\mu} \sin\lambda - m \mu^2 \cos\lambda - k_x \mu \sin\lambda \quad (4.2.5.19)$$

$$\mathbf{u}_y = m \dot{\mu} \cos\lambda - m \mu^2 \sin\lambda + k_y \mu \cos\lambda \quad (4.2.5.20)$$

Now let the torque bounds be  $-T \leq \mathbf{u}_x, \mathbf{u}_y \leq +T$ . Then the bounds on  $\dot{\mu}$  are

$$\frac{-T - m \mu^2 \cos\lambda - k_x \mu \sin\lambda}{m \sin\lambda} \leq \dot{\mu} \leq \frac{+T - m \mu^2 \cos\lambda - k_x \mu \sin\lambda}{m \sin\lambda} \quad (4.2.5.21)$$

and

$$\frac{-T + m \mu^2 \sin\lambda - k_y \mu \cos\lambda}{m \cos\lambda} \leq \dot{\mu} \leq \frac{+T + m \mu^2 \sin\lambda - k_y \mu \cos\lambda}{m \cos\lambda} \quad (4.2.5.22)$$

The admissible region consists of the region where the inequalities given above allow some value of the acceleration  $\dot{\mu}$ , as previously described. Simplifying the resulting inequalities gives the admissible region as that area of the phase plane where

$$m \mu^2 + (k_x - k_y) \mu \sin\lambda \cos\lambda + T(\sin\lambda + \cos\lambda) \geq 0 \quad (4.2.5.23)$$

and

$$-m \mu^2 - (k_x - k_y) \mu \sin\lambda \cos\lambda + T(\sin\lambda + \cos\lambda) \geq 0 \quad (4.2.5.24)$$

Using the values  $m=2$ ,  $k_x=0$ ,  $k_y=10$ , and  $T=\sqrt{2}$  gives the region plotted in Figure 4.2.11 and clearly shows the island.

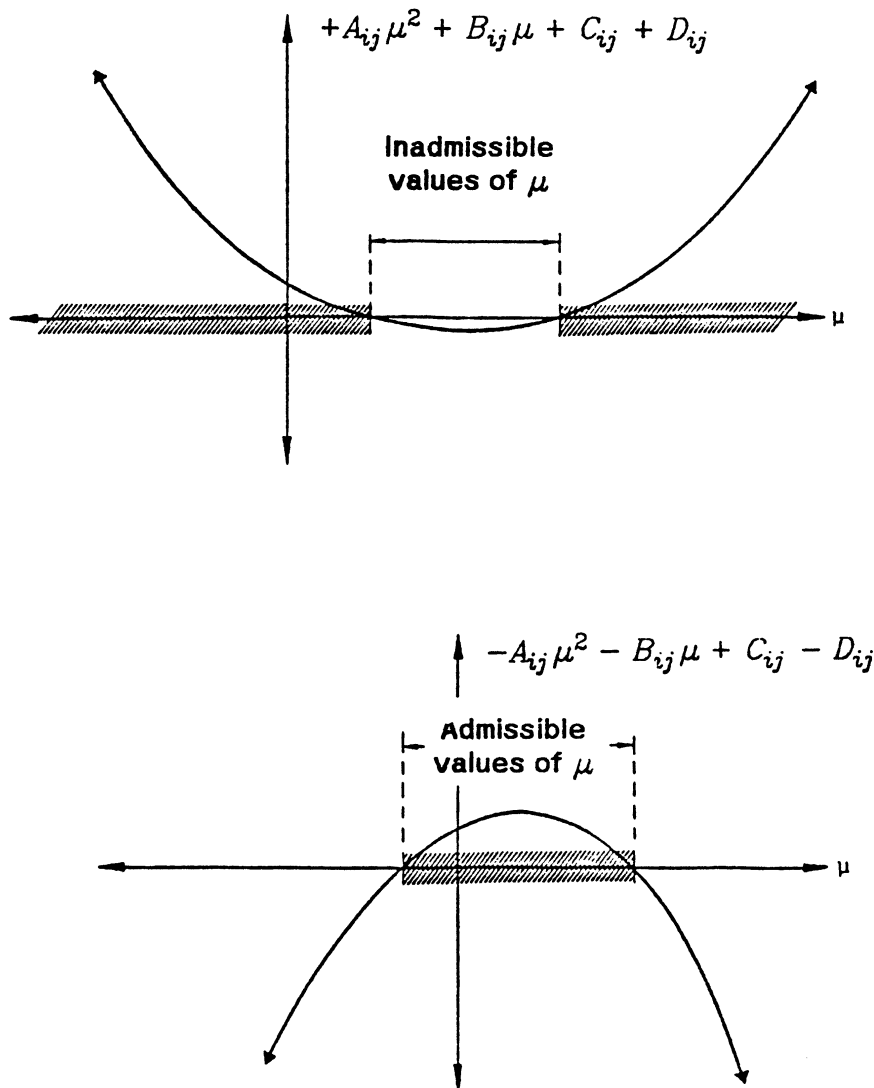


Figure 4.2.1. Admissible regions of  $\mu$  determined by parabolic constraints

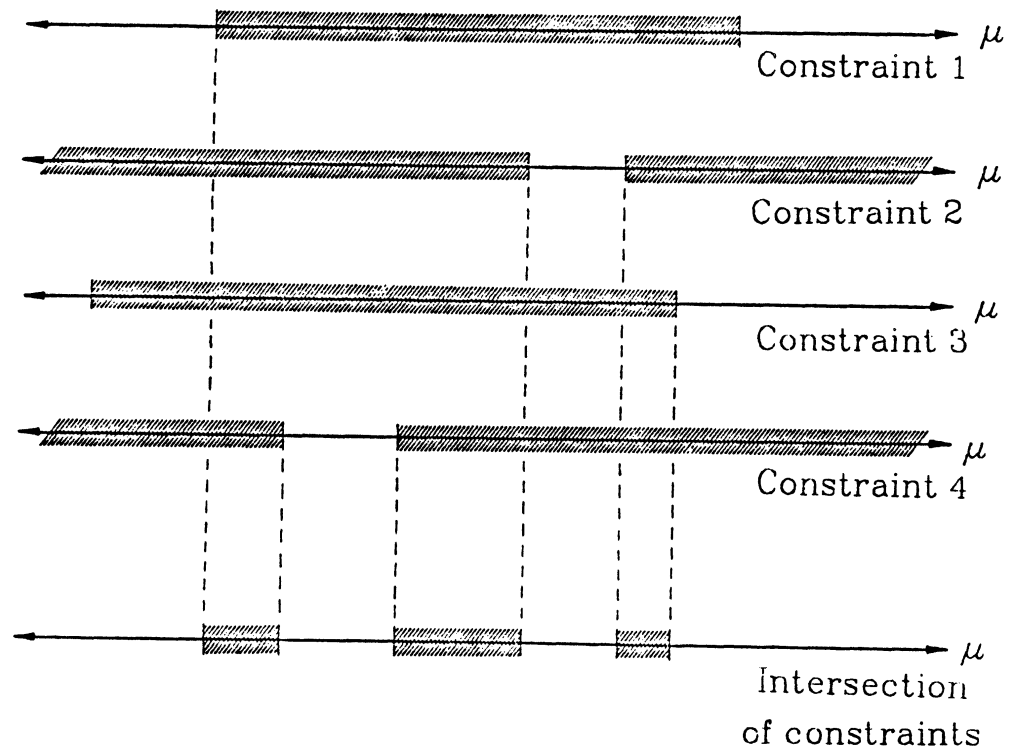
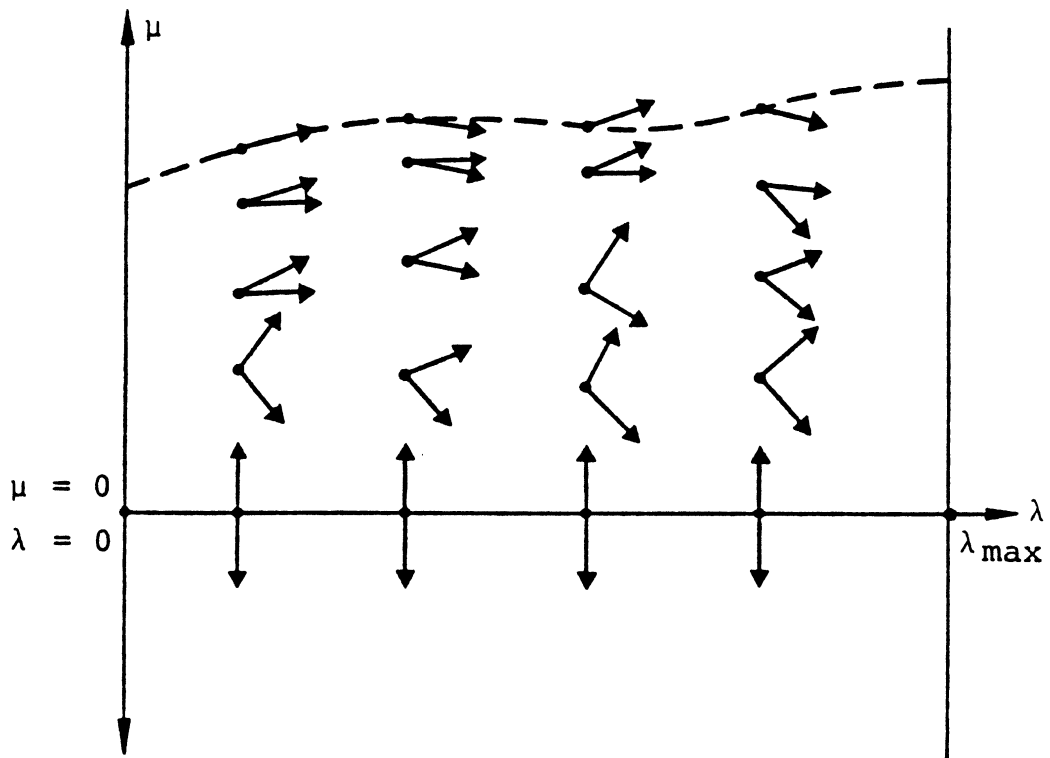


Figure 4.2.2. Intersection of admissible regions of  $\mu$





Upper arrows give maximum accerleration  
 Lower arrows give maximum deceleration  
 Dashed curve is boundary of admissible region

Figure 4.2.3. Acceleration and deceleration vectors at each phase point

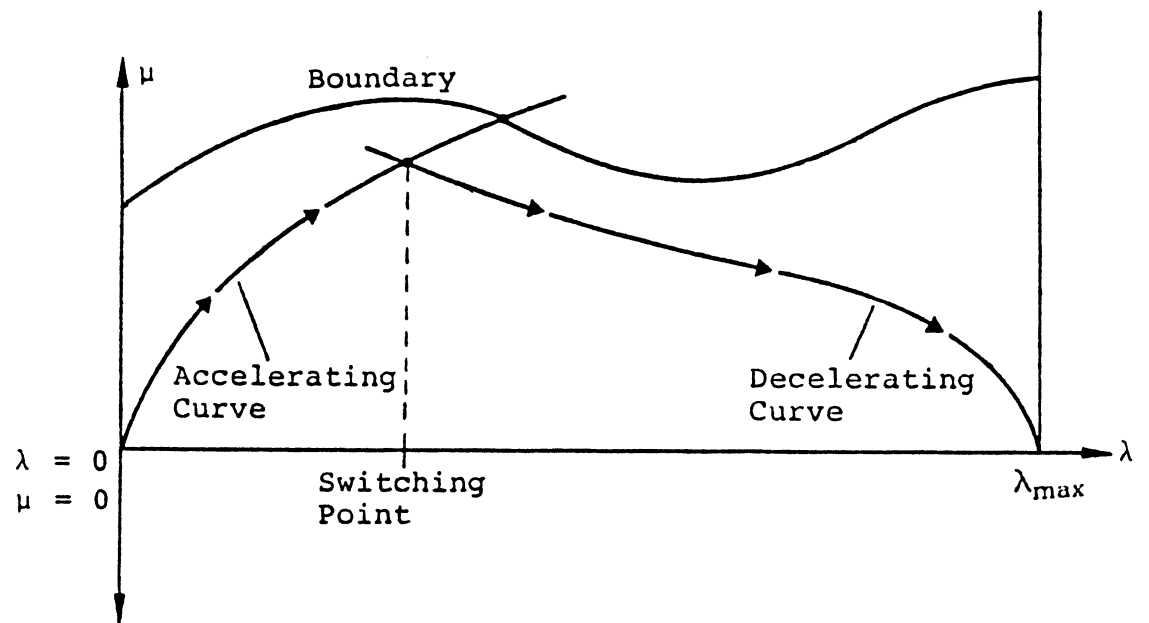


Figure 4.2.4. Case when accelerating and decelerating curves intersect

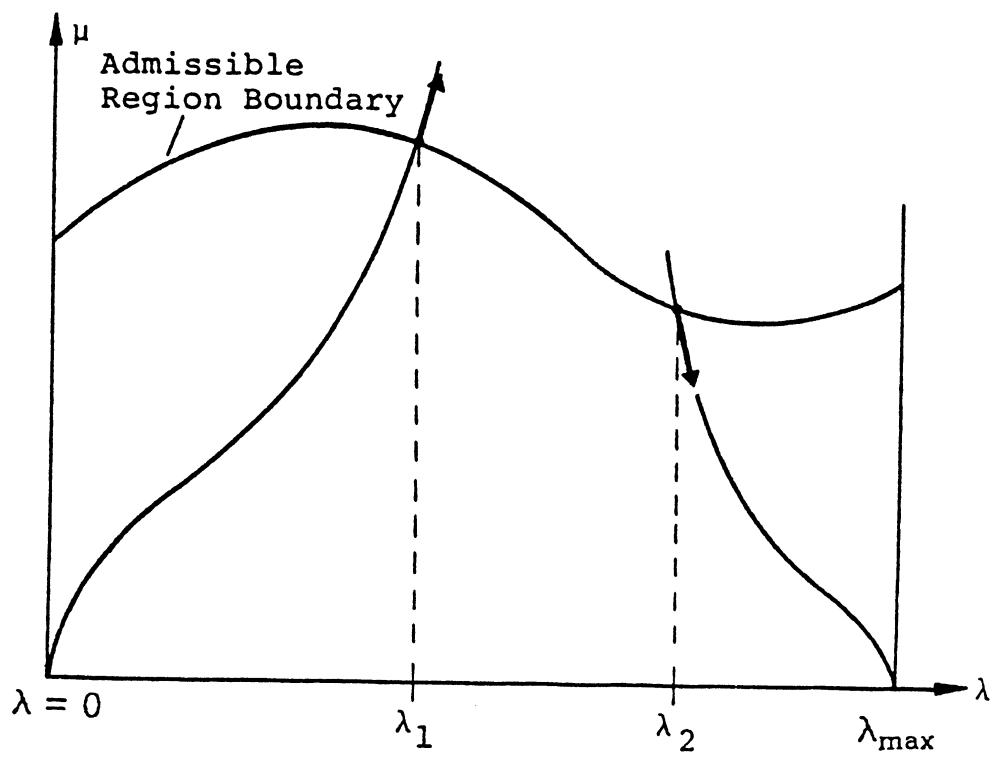
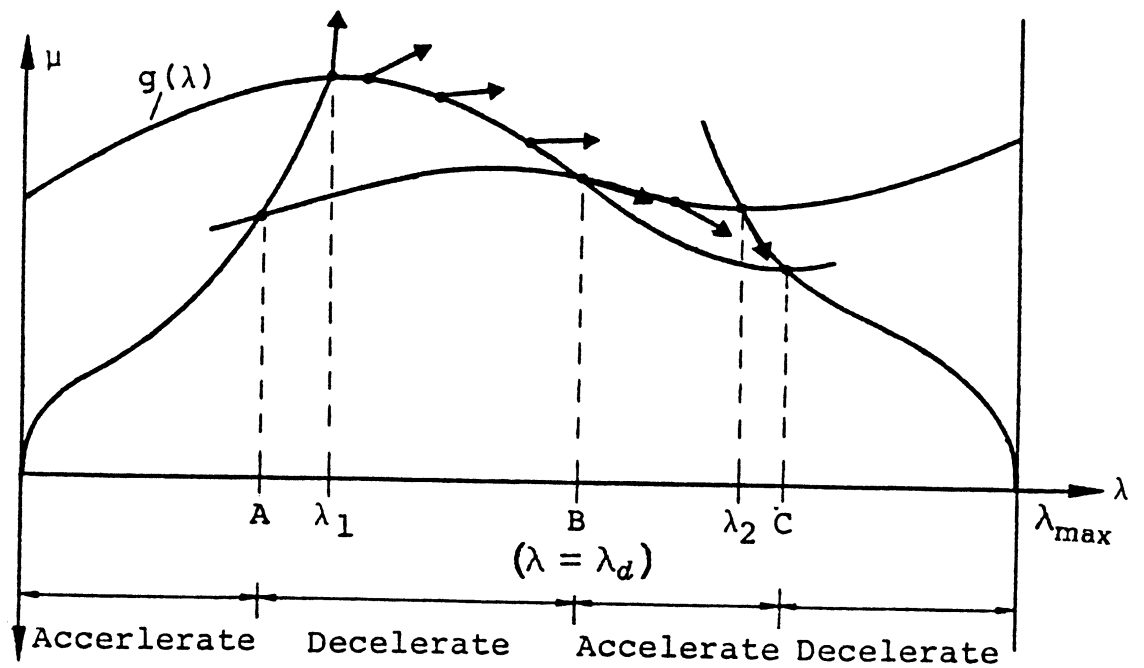


Figure 4.2.5. Case when accelerating and decelerating curves do not intersect



A, B, and C are switching points  
 B is a point of osculation between  
 $g(\lambda)$  and the trajectory

Figure 4.2.6. A complete optimal trajectory with three switching points

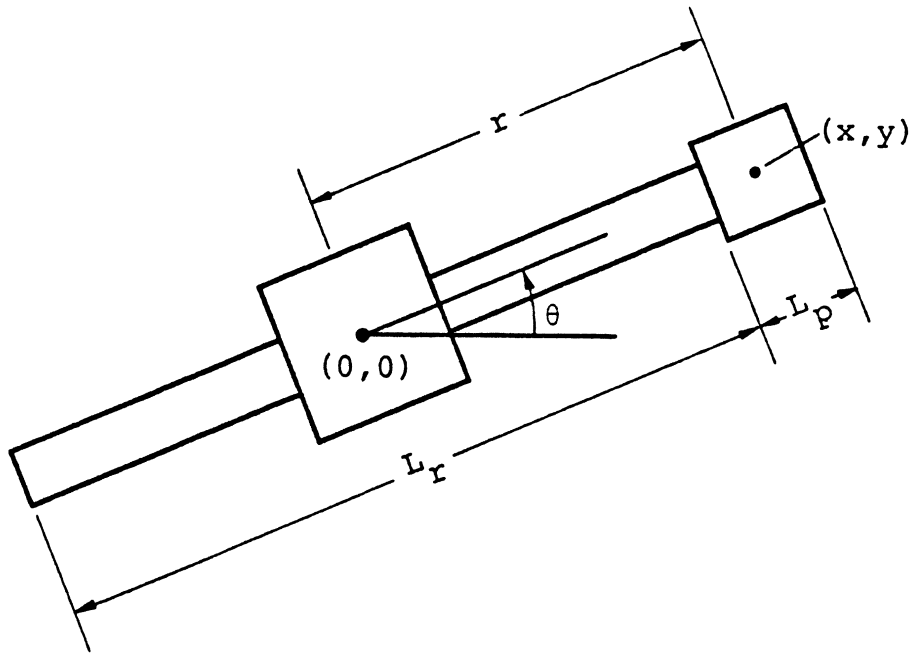
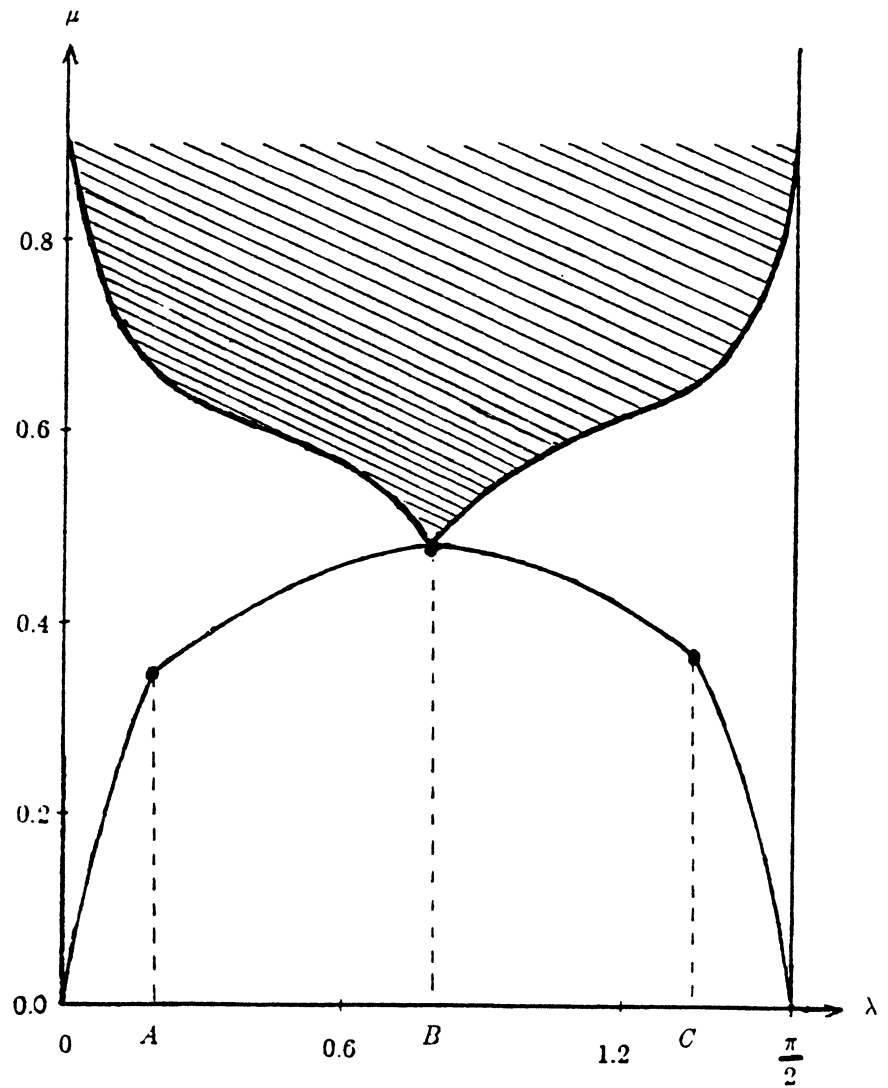



Figure 4.2.7. The two degree-of-freedom polar robot



— Boundary curve  
 — Optimal trajectory  
 Inadmissible region

A,B,C : Switching points

Figure 4.2.8. Optimal trajectory, zero friction case

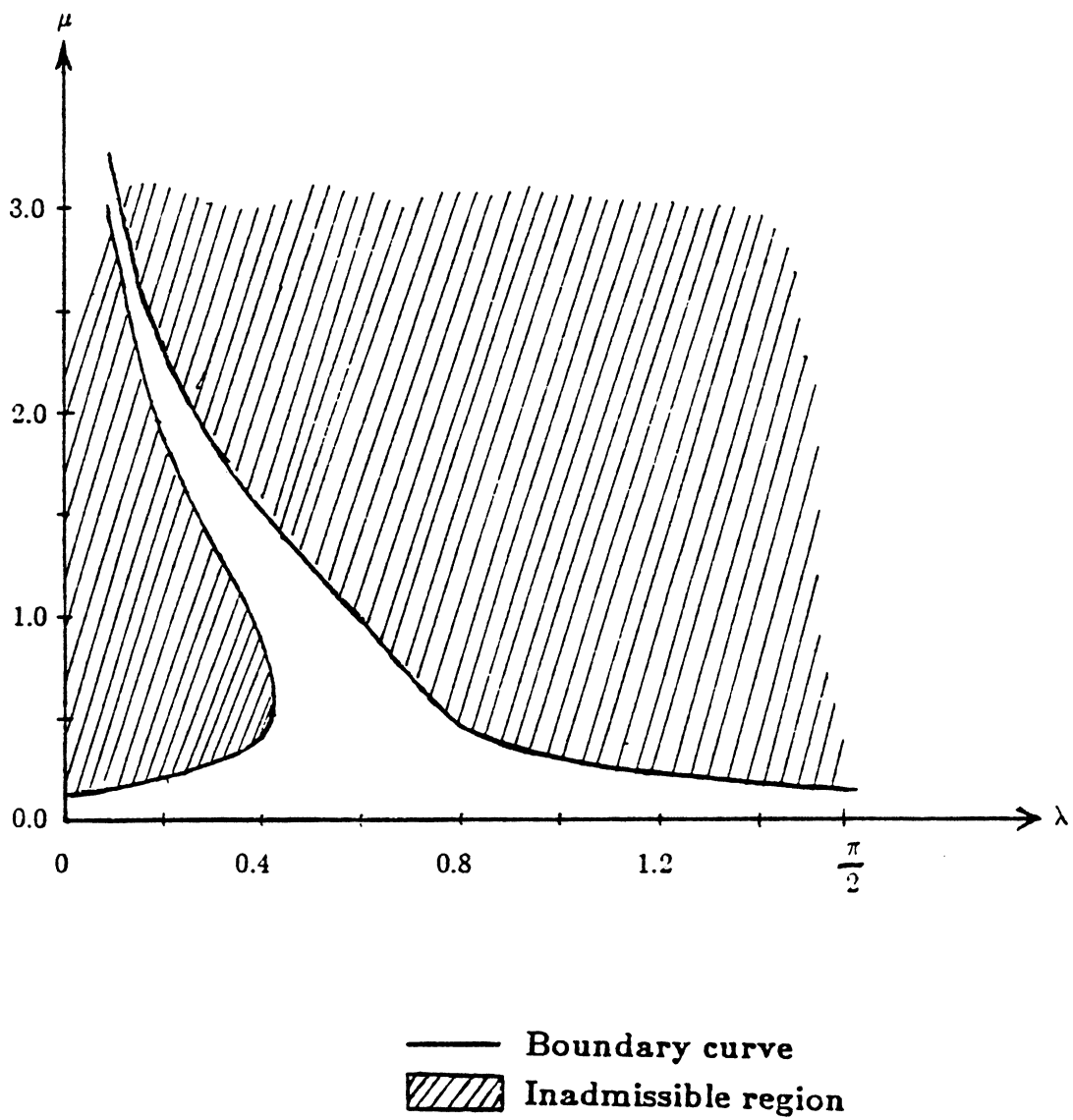
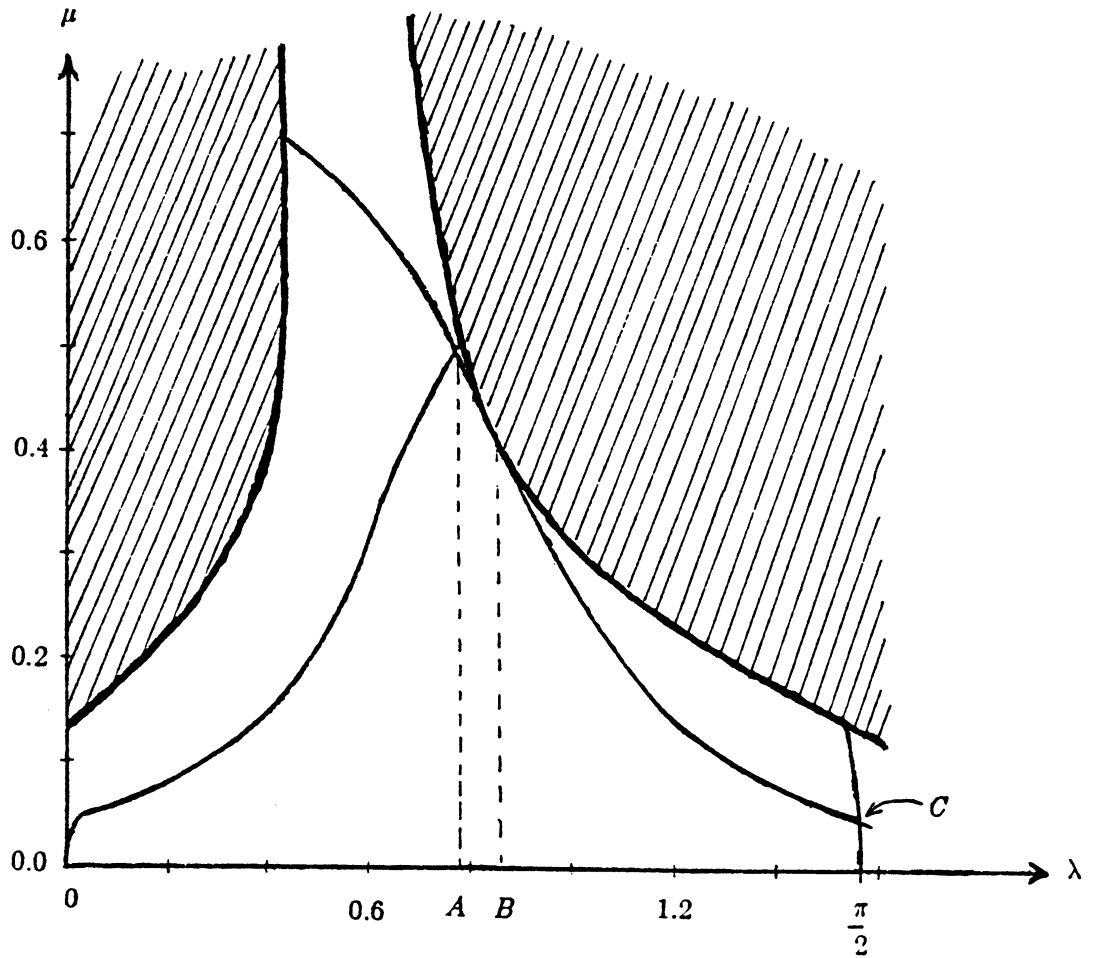





Figure 4.2.9. Admissible region, high-friction case



 Boundary curve  
 Optimal trajectory  
 Inadmissible region

A,B,C : Switching points

Figure 4.2.10. Optimal trajectory, high-friction case



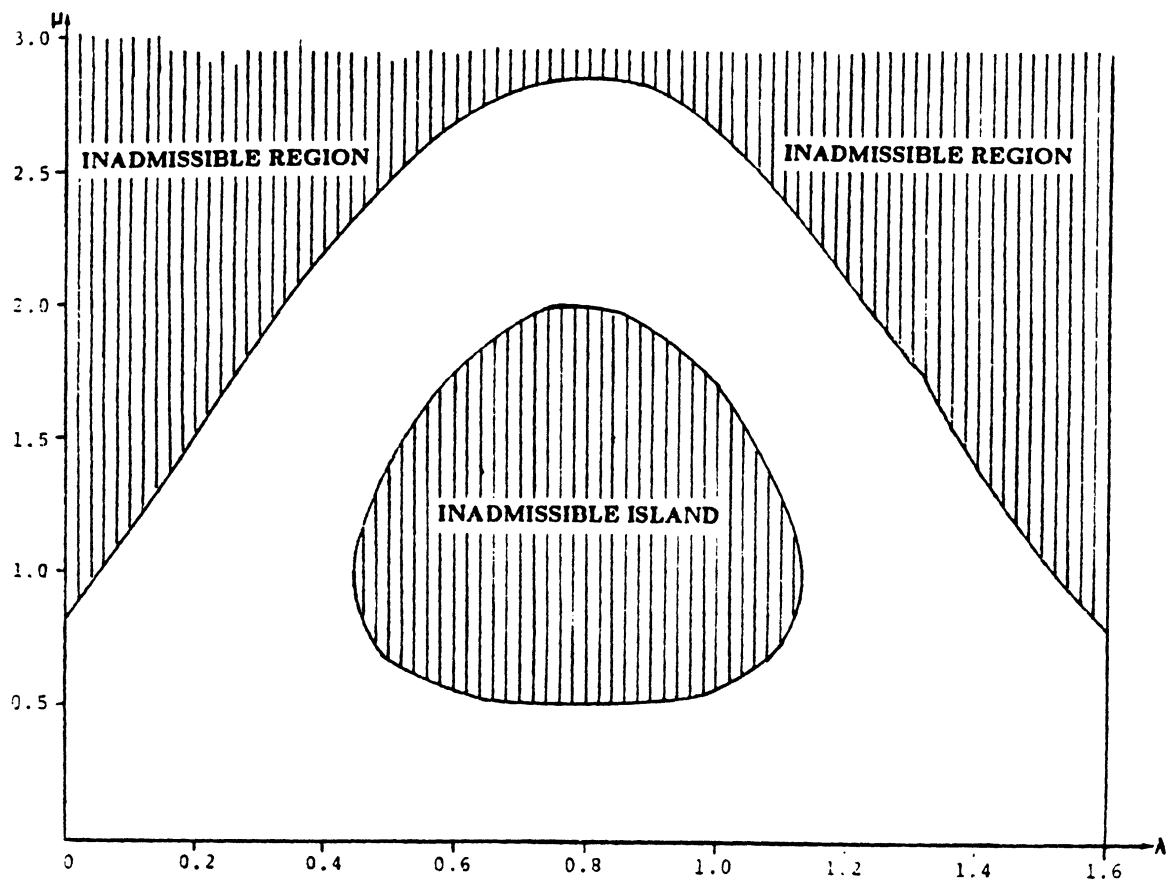


Figure 4.2.11. Admissible region for cartesian manipulator

### 4.3. The Dynamic Programming Method

The algorithms presented in the previous section are adequate for solving the minimum-time trajectory planning problem, provided that the actuator torque limits do not depend on the joint velocity in a complicated way and are independent of one another. However, in situations where driving the robot consumes large amounts of power, the assumption that minimum time is equivalent to minimum cost may not be valid; then the phase plane algorithm gives a solution to the wrong problem. Interdependence of actuator torque constraints is another very real possibility that the phase plane method cannot handle. This interdependence may occur, for example, when a robot uses a common power supply for the servo amplifiers for all joints, or when all joints of a hydraulic robot are driven from a common pump. Finally, it is assumed that the joint torques can be changed instantaneously. This is only approximately true, and indeed it is sometimes desirable to limit the derivatives of the joint torques (or, equivalently, the jerk, or derivative of the acceleration) to prevent excessive mechanism wear.

One means of eliminating these limitations is to use a more general optimization technique. The method proposed here is to use dynamic programming [15] to find the optimal phase plane trajectory. The dynamic programming technique places few restrictions on the cost function that is to be minimized. Putting limits on jerk is also (theoretically) possible, and interdependence of torque bounds can be handled fairly painlessly, as will be seen later. One of the major drawbacks of dynamic programming, the "curse of dimensionality", is not an issue in the trajectory planning problems considered here, since the use of the parametric functions (4.1.1) reduces the dimension of the state space from  $2n$  for an  $n$ -jointed manipulator to two.

### 4.3.1. Problem Formulation

As before, the manipulator is assumed to move along a path given by (4.1.1), and the dynamics are written in the form of (4.1.6). Initially, it is also assumed that the set of realizable torques can be given in terms of the state of the system, i.e., in terms of the robot's position and velocity. Then we have

$$\mathbf{u} = (u_1, u_2, \dots, u_n)^T \in E(\mathbf{q}, \dot{\mathbf{q}}) \quad (4.3.1.1)$$

where  $\dot{\mathbf{q}}$  is the first derivative of  $\mathbf{q}$  with respect to time, and  $u_i$  is the  $i^{\text{th}}$  actuator torque/force.  $E$  is a function from  $R^n \times R^n$  to the space of sets in  $R^n$ . In other words, given the position and velocity,  $E$  determines a set in the input space. The input torques  $u_i$  are *realizable* for position  $\mathbf{q}$  and velocity  $\dot{\mathbf{q}}$  if and only if the torque vector  $\mathbf{u}$  is in the set  $E(\mathbf{q}, \dot{\mathbf{q}})$ . Note that independence of the actuator torque limits is *not* assumed.

If limits on the derivatives of the joint torque (or, equivalently, derivative of the acceleration, or the  *jerk*) are also to be applied, then we also must satisfy the inequalities

$$|\dot{u}_i| \leq K_i(\lambda, \mu) \quad (4.3.1.2)$$

The cost  $C$  given by (3.3) may be transformed into

$$C = \int_0^{\lambda_{\max}} L(\lambda, \mu, \mathbf{u}_i) d\lambda \quad (4.3.1.3)$$

The minimum-cost trajectory planning problem then becomes that of finding  $\mu = \mu^*(\lambda)$  which minimizes (4.3.1.3) subject to Eq. (4.1.6) and the inequalities (4.3.1.1) and (4.3.1.2).

### 4.3.2. Trajectory Planning Using Dynamic Programming

To see how dynamic programming can be applied to this problem, first note that by using the parameterized path (4.1.1), the dimensionality of the problem has been reduced; there will be only two state variables  $\lambda$  and  $\mu$ , *regardless* of how many joints the robot has. The "curse of dimensionality" has therefore been avoided. To apply dynamic programming, one first must divide the phase plane into a discrete grid. Then, the cost of going from one point on the grid to the next must be calculated. Note that since  $u_i$  will be determined as a function of  $\lambda$  and  $\mu$ , Eq. (4.3.1.3) can be written strictly in terms of  $\lambda$  and  $\mu$ ; thus the cost computation can be done entirely in phase coordinates. Once costs have been computed, the usual dynamic programming algorithm can be applied, and positions, velocities and torques can be obtained from the resulting optimal trajectory and Eqs. (4.1.1) and (4.1.6).

The informal description given above describes the general approach to the MCTP problem. In detail, there are some complications. Therefore, some simplifying but realistic assumptions will be made as we proceed. First, the grid's  $\lambda$ -divisions are assumed to be small enough so that the functions  $M_i$ ,  $Q_i$ ,  $R_i$ ,  $S_i$ , and  $\frac{df^i}{d\lambda}$  do not change significantly over a single  $\lambda$ -interval. Then, the coefficients of Eq. (4.1.6) are effectively constant. So are the coefficients of (4.2.3.1), which we will use as our (single) dynamic equation. (Other assumptions, such as piecewise linearity, are possible. However, these assumptions complicate the analysis considerably.) Note that (4.2.3.1) does not explicitly depend on time. Therefore, for purposes of carrying out the dynamic programming algorithm, we may treat the quantities  $\lambda$  and  $\mu$  as a stage variable and a single state variable rather than two state variables. Using (4.2.3.1) as our (single) dynamic equation, and noting that  $M$ ,  $Q$ ,  $R$ , and  $S$  are

approximately constant over one  $\lambda$ -interval, we need to find a solution to (4.2.2.2) which meets the boundary conditions

$$\mu(\lambda_k) = \mu_0, \quad \mu(\lambda_{k+1}) = \mu_1 \quad (4.3.2.1)$$

in the interval  $[\lambda_k, \lambda_{k+1}]$  (see Figure 4.3.1). In order to do this, some form for the inputs  $u_i$  needs to be chosen. It should be noted that as the DP grid becomes finer, the precise form of the curves joining the points of the grid matters less. As long as the curves are smooth and monotonic, the choice of curves makes a smaller and smaller difference as the grid shrinks. The implication of this is that we may choose virtually any curve that is convenient, and as long as the grid size is small, the results should be a good approximation to the optimal trajectory.

We will use the form

$$u_i = Q_i \mu^2 + R_i \mu + V_i \quad (4.3.2.2)$$

for the input, where the  $V_i$  are constants that may be chosen to make the solution meet the boundary conditions (4.3.2.1). Form (4.3.2.2) was chosen because it yields particularly simple solutions.

In what follows, we first obtain a solution without torque bound interaction, and then extend the solution to accommodate torque constraints of a much more general type.

### 4.3.3. Case of Non-Interacting Torque Bounds

When the joint torque bounds do not interact, the sets  $E$  in (4.3.1.1) are given by

$$E(\mathbf{q}, \dot{\mathbf{q}}) = \left\{ (\mathbf{u}_1, \dots, \mathbf{u}_n)^T \mid \mathbf{u}_{\min}^i(\mathbf{q}, \dot{\mathbf{q}}) \leq \mathbf{u}_i \leq \mathbf{u}_{\max}^i(\mathbf{q}, \dot{\mathbf{q}}) \right\} \quad (4.3.3.1)$$

Taking the projection of the input torque vector, as given by (4.3.2.2), onto the velocity vector  $\frac{df^i}{d\lambda}$  gives  $U = Q\mu^2 + R\mu + V$ , where  $V = V_i \frac{df^i}{d\lambda}$ . Plugging this into the differential Eq. (4.2.3.1) gives

$$M \frac{d\mu}{d\lambda} + Q\mu + R + \frac{1}{\mu} (S - Q\mu^2 - R\mu - V) = 0 \quad (4.3.3.2)$$

or

$$\frac{d\mu}{d\lambda} = -\frac{1}{\mu} \frac{(S - V)}{M}. \quad (4.3.3.3)$$

Solving this equation, we have

$$\lambda = K - \frac{M}{2(S - V)} \mu^2. \quad (4.3.3.4)$$

Evaluating the constant of integration  $K$  and the constant  $V$  so that (4.3.3.4) meets the boundary conditions (4.3.2.1), one obtains

$$\lambda = \frac{\lambda_k (\mu_1^2 - \mu^2) + \lambda_{k+1} (\mu^2 - \mu_0^2)}{\mu_1^2 - \mu_0^2}. \quad (4.3.3.5)$$

Solving for  $\mu$  in terms of  $\lambda$  gives

$$\mu = \sqrt{\frac{(\lambda_{k+1} - \lambda)\mu_0^2 + (\lambda - \lambda_k)\mu_1^2}{\lambda_{k+1} - \lambda_k}} \quad (4.3.3.6)$$

Now that the path is known over one  $\lambda$ -interval, we need to know the inputs  $u_i$  and the components of the incremental cost.

To evaluate the input torques, we may use Eq. (4.1.6) and the value of  $\dot{\mu}$ . Noting that  $\dot{\mu} \equiv \frac{\dot{\mu}}{\mu} \cdot \mu \equiv \mu \frac{d\mu}{d\lambda}$  and using Eq. (4.3.3.3), we obtain

$$\dot{\mu} = \frac{(V-S)}{M} = \text{constant}. \quad (4.3.3.7)$$

The quantities  $M$  and  $S$  are given, and, using Eqs. (4.3.3.6) and (4.3.3.7),  $V$  can be calculated to be

$$V = S + \frac{M}{2} \cdot \frac{\mu_1^2 - \mu_0^2}{\lambda_{k+1} - \lambda_k} \quad (4.3.3.8)$$

which gives

$$\dot{\mu} = \frac{\mu_1^2 - \mu_0^2}{2(\lambda_{k+1} - \lambda_k)}. \quad (4.3.3.9)$$

Therefore, the equations for  $u_i$  become

$$u_i = Q_i \mu^2 + R_i \mu + S_i + M_i \cdot \frac{\mu_1^2 - \mu_0^2}{2(\lambda_{k+1} - \lambda_k)}. \quad (4.3.3.10)$$

Assuming the joint torque limits are independent, determining whether joint  $i$  ever demands any unrealizable torques requires that we know the maximum and minimum values of  $u_i$  over the interval  $[\lambda_k, \lambda_{k+1}]$  (or equivalently over the interval  $[\min(\mu_0, \mu_1), \max(\mu_0, \mu_1)]$ , since  $\lambda$  is a monotonic function of  $\mu$  over the interval under consideration). The maxima/minima may occur at one of three  $\mu$  values, namely  $\mu_0$ ,

$\mu_1$ , and that value of  $\mu$  that maximizes or minimizes  $u_i$  over the unrestricted range of  $\mu$ . In the latter case, the value of  $\mu$  is

$$\mu_m = -\frac{R_i}{2Q_i}. \quad (4.3.3.11)$$

If the condition

$$\min(\mu_0, \mu_1) \leq \mu_m \leq \max(\mu_0, \mu_1) \quad (4.3.3.12)$$

holds, then the point  $\mu_m$  needs to be tested. Otherwise the torques must be computed and checked only at the endpoints of the interval. (If  $M_i$ ,  $Q_i$ ,  $R_i$ , and  $S_i$  were assumed to be piecewise linear in  $\lambda$ , then the formula analogous to (4.3.3.10) would be a quartic rather than a quadratic, and in theory three "midpoints", found by solving a cubic, would have to be tested. But as a practical matter, testing the endpoints of the interval is probably adequate.)

Given the formulae for the velocity and the joint torques, the incremental cost can be found using the formula

$$C = \int_{\lambda_t}^{\lambda_{t+1}} L(\lambda, \mu, \mathbf{u}_i) d\lambda \quad (4.3.3.13)$$

where  $\mu$  and  $\mathbf{u}_i$  are given as functions of  $\lambda$  by formulae (4.3.3.6) and (4.3.3.10), respectively. It may be possible to evaluate this integral directly; if not, then the integral may be approximated by any of the standard techniques. Section 4.3.8 shows that the DP algorithm converges when the integral is approximated using the Euler method. Using more sophisticated algorithms should give faster convergence than the Euler method.



With these formulae at hand, it is now possible to state the dynamic programming algorithm in detail. Initially, the algorithm will be stated for the case in which there are no limits on the time derivatives of the torques. These constraints will be considered later in Section 4.3.5. The algorithm, given the dynamic equations (4.1.6), the equations of the curve (4.1.1), the joint torque constraints (4.3.3.1), and the incremental cost (4.3.3.13), is:

- S1. Determine the derivatives  $\frac{df^i}{d\lambda}$  of the parametric functions  $f^i(\lambda)$ , and from these quantities and the dynamic equations determine the coefficients of Eqs. (4.1.6).
- S2. Divide the  $(\lambda, \mu)$  phase plane into a rectangular grid with  $N_\lambda$  divisions on the  $\lambda$ -axis and  $N_\mu$  divisions on the  $\mu$ -axis. Associate with each point  $(\lambda_m, \mu_n)$  on the grid a cost  $C_{mn}$  and a "next row" pointer  $P_{mn}$ . Set all costs  $C_{mn}$  to infinity, except for the cost of the desired final state, which should be set to zero. Set all the pointers  $P_{mn}$  to null, i.e., make them point nowhere. Set the column counter  $\alpha$  to  $N_\lambda$ .
- S3. If the column counter  $\alpha$  is zero, then stop.
- S4. Otherwise, set the current-row counter  $\beta$  to 0.
- S5. If  $\beta = N_\mu$ , go to S12.
- S6. Otherwise, set the next-row counter  $\gamma$  to 0.
- S7. If  $\gamma = N_\mu$ , go to S11.
- S8. For rows  $\beta$  and  $\gamma$ , generate the curve that connects the  $(\alpha-1, \beta)$  entry to the  $(\alpha, \gamma)$  entry, as in Figure 4.3.2. For this curve, test, as described in the previous paragraphs, to see if the required joint torques are in the range given by

inequalities (4.3.3.1). If they are not, go to S10.

- S9. Compute the cost of the curve by adding the cost  $C_{\alpha\gamma}$  to the incremental cost of joining point  $(\alpha-1,\beta)$  to point  $(\alpha,\gamma)$ . If this cost is less than the cost  $C_{\alpha-1,\beta}$ , then set  $C_{\alpha-1,\beta}$  to this cost, and set the pointer  $P_{\alpha-1,\beta}$  to point to that grid entry  $(\alpha,\gamma)$  that produced the minimum cost, i.e. set  $P_{\alpha-1,\beta}$  to  $\gamma$ .
- S10. Increment the next-row counter  $\gamma$  and go to S7.
- S11. Increment the current-row counter  $\beta$  and go to S5.
- S12. Decrement the column counter  $\alpha$  and go to S3.

Finding the optimal trajectory from the grid is then a matter of tracing the pointers  $P_{mn}$  from the initial to the final state. If the first pointer is null, then no solution exists; otherwise, the successive grid entries in the pointer chain give the optimal trajectory. Given the optimal trajectory, it is then possible to calculate joint positions, velocities, and torques.

#### 4.3.4. Case of Interacting Torque Bounds

It has been assumed in the preceding discussion that the joint torque limits do not interact, i.e., that increasing the torque on one joint does not decrease the available torque at another joint. This assumption manifests itself in the form of the torque constraint inequalities (4.3.3.1). This assumption is probably correct in many cases, but in others it certainly is not. Consider, for example, a robot that has a common power supply for the servo amplifiers for all joints. The power source will have some finite limit on the power it can supply, so that the sum of the power consumed by all the joints must be less than that limit. A similar situation arises when

a single pump drives several hydraulic servoes. The pump will have finite limits on both the pressure and the volume flow it can produce. In fact, it may be desirable to have torque limits interact, in the sense that using a single power source to drive the robot may cost less than using several independent power sources. In most cases multiple power sources would not be used at their maximum capacities simultaneously, so that there would be little to be gained by having independent power sources, while the cost of a single large power source would quite possibly be considerably less than that of several small ones.

Because of the possibility of torque limit interaction, it will be assumed here that the inequalities (4.3.3.1) are replaced with the constraint (4.3.1.1), namely  $(\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n)^T \in E(\mathbf{q}, \dot{\mathbf{q}})$ . It is interesting to note that the limits described in the previous paragraph all produce set functions  $E$  in Eq. (4.3.1.1) which are *convex*. For example, if the sum of the power consumed (or produced) in all the joints is bounded, one obtains the bounds

$$P_{\min} \leq \mathbf{u}_i \dot{\mathbf{q}}^i \leq P_{\max}. \quad (4.3.4.1)$$

For any given velocity, this is just the region between a pair of parallel hyperplanes in the joint space. Likewise, for independent torque bounds, the realizable torques are contained in a hyper-rectangular prism, another convex region. Since the intersection of any number of convex sets is a convex set, any combination of these constraints will also yield a convex constraint set. In this light, it is reasonable to make the assumption that the set  $E(\mathbf{q}, \dot{\mathbf{q}})$  is convex. This assumption is important in the analysis that follows.

To see how we may make use of this convexity condition, consider the test for realizability of torques used in the method presented thus far. This test made explicit use of the assumption that the torque bounds do not interact. In order to handle interacting torque bounds using an approach like that of Section 4.3.3, it must be possible to determine whether *all* torques are realizable over any given  $\lambda$ -interval. If the torques have the form used in Eq. (4.3.2.2), then this is in general not possible with any finite number of tests; even in the two-dimensional case, the torques trace out conics in the input space, and there is no general way to determine whether a segment of a conic is entirely contained within a convex set.

Though the question of whether a set of torques is realizable cannot in general be given a definite answer, the realizability question can be answered in some cases. To see how this can be done, consider again the tests for realizability previously described. The maximum and minimum torques for each joint are determined, and these torques are checked. While Eq. (4.3.2.2) describes a curve in the joint torque space, the individual torque limits describe a box-shaped volume. The curve describing the joint torques will be entirely contained inside this box. Thus if every point in the box is admissible, then so is every point on the curve. This "reduces" the problem of determining whether every point of a one-dimensional set is realizable to the problem of determining whether every point of a higher-dimensional set is realizable. However, this higher-dimensional set has a special shape; it is a convex polyhedron, and will be contained in the (convex) set  $E$  if and only if all its vertices are in  $E$ . Thus by testing a finite number of points, the question of whether a particular set of torques is realizable may sometimes be given a definite "yes" answer.

If this test does not give a definite answer, then the set of inputs in question must be discarded, even though that set may in fact be realizable. However, as the grid size shrinks, the size of the bounding box for the torques also shrinks, so that in the limit the test becomes a test of a single point. Therefore as the grid shrinks, the percentage of valid torques thrown away approaches zero, and the optimal solution will be found.

This method of handling interacting torque bounds requires only one change in the DP algorithm. Step S8, which checks to see if the torques are realizable, must be replaced with a step that generates all corners of the bounding box and tests these points for realizability. If any of the corners does not represent a realizable set of torques, then the test fails. Thus we have

S8'. For rows  $\beta$  and  $\gamma$ , generate the curve that connects the  $(\alpha-1, \beta)$  entry to the  $(\alpha, \gamma)$  entry, as in Figure 4.3.2. For this curve, generate the maximum and minimum torques at each joint. Check each torque  $n$ -tuple formed from the maximum and minimum joint torques. (These are the corners of the bounding box.) If any of these  $n$ -tuples are not contained in the set  $E$ , then go to S10.

#### 4.3.5. Accommodation of Jerk Constraints

The methods described thus far have ignored the jerk constraints (4.3.1.2) which limit the derivatives of the joint torques. Taking these limits into account effectively requires that a third state variable be added. That variable can be taken to be the pseudo-acceleration  $\dot{\mu}$ , say  $\nu \equiv \dot{\mu}$ . Differentiating the equation for the torque, one obtains

$$\dot{u}_i = \dot{M}_i \nu + M_i \dot{\nu} + \dot{Q}_i \mu^2 + 2Q_i \mu \dot{\nu} + \dot{R}_i \mu + R_i \dot{\nu} + \dot{S}_i. \quad (4.3.5.1)$$

Using the identity  $\frac{d\phi}{dt} = \frac{d\phi}{d\lambda} \frac{d\lambda}{dt} = \frac{d\phi}{d\lambda} \mu$ , this equation becomes

$$\dot{u}_i = M_i \dot{\nu} + \left( \frac{dM_i}{d\lambda} + 2Q_i \right) \mu \nu + R_i \nu + \frac{dQ_i}{d\lambda} \mu^3 + \frac{dR_i}{d\lambda} \mu^2 + \frac{dS_i}{d\lambda} \mu \quad (4.3.5.2)$$

If there are no jerk constraints, then the parameter  $\dot{\mu}$  in Eq. (4.1.6) can be manipulated as needed. When there are jerk constraints, we must instead manipulate  $\dot{\nu}$  in Eq. (4.3.5.2). Eq. (4.3.5.2) and constraints (4.3.1.2) then give constraints on  $\dot{\nu}$ , just as Eq. (4.1.6) and constraints (4.3.1.1) yield constraints on  $\dot{\mu}$ .

To solve the optimization problem with jerk constraints using dynamic programming, a three-dimensional grid is required, with one dimension for each of  $\lambda$ ,  $\mu$ , and  $\nu$ . Some form must be assumed for the "inputs"  $\dot{u}_i$ , as was done for  $u_i$  when there were no jerk constraints, i.e., Eq. (4.3.2.2). Because the grid points that the DP algorithm must join form a pair of planes, rather than a pair of lines or columns, as in the two-dimensional case, the form of the input must contain two arbitrary constants instead of one. If only one parameter is used, then it will not be possible to connect arbitrarily chosen points in the DP grid. The problem is thus inherently more complicated than the two-dimensional case, at least in terms of the algebra required to produce a solution. The procedure is otherwise the same as that for the two-dimensional case.

Because of the algebraic complexity and because of the computer time that would be consumed by a three-dimensional dynamic programming algorithm, no examples were worked for this case. However, section 4.4 presents a minimum time

algorithm which handles jerk constraints in a much more satisfactory manner.

#### 4.3.6. Algorithm Complexity

The usefulness of the dynamic programming technique depends on its being reasonably efficient in terms of use of computing resources, i.e., it must run reasonably fast and must not use too much memory. Since the trajectory planning is done off-line, the algorithm's time requirements are not particularly critical; nevertheless, the time required must not be exorbitant if trajectory planning is to be worthwhile. Likewise, computer memory is relatively inexpensive, but nevertheless puts some limits on the accuracy with which the dynamic programming algorithm can be performed. In this section we present an approximate analysis of the time and memory requirements of the algorithm. Of course, precise numbers will depend rather heavily upon such variables as the computer on which the algorithm is to run, the language in which it is implemented, the compiler used, and the skill of the programmer who writes the code, so the expressions derived here contain a number of implementation-dependent constants.

It is easy to compute the storage requirements for the algorithm. The memory allotted to the program itself is essentially fixed. The size of the grid used for the dynamic programming algorithm varies with the fineness of the grid and the amount of storage required per point on the grid. The grid has  $N_\mu$  rows and  $N_\lambda$  columns. Each entry must contain a cost  $C$  and a pointer  $P$ . The size of an entry will then be

$$GS = S_c + S_p \tag{4.3.6.1}$$

where  $GS$  is the storage requirement for a single point of the grid and  $S_c$  and  $S_p$  are the amounts of storage required to record the cost and the pointer to the next row, respectively. In the implementation presented here, parameterized curves are represented as arrays of points. If one assumes that there is one point per  $\lambda$ -division, then there is an additional  $S_d + N_\lambda S_i$ , where  $S_i$  is the storage required for one interpolation point on the curve and  $S_d$  is a certain fixed storage per curve. Multiplying  $GS$  by the number of grid entries and adding the amount of storage  $PS$  required for the program and the storage required for the curve gives total storage  $TS$  as

$$TS = PS + N_\lambda N_\mu (S_c + S_p) + N_\lambda S_i + S_d. \quad (4.3.6.2)$$

For the numerical example presented in this paper, all arithmetic was done in double precision, and integers and pointers are four bytes long. Then for a six-jointed arm the storage required is, ignoring the program storage,

$$TS = 12N_\lambda N_\mu + 80 + 448N_\lambda \quad (4.3.6.3)$$

For example, a  $20 \times 80$  grid requires 28,240 bytes. This can, of course, be reduced considerably by using single rather than double precision; however, even using double precision, the storage required is generally available on small microprocessors.

Calculating the time required to perform the dynamic programming algorithm is somewhat more difficult. There will be  $N_\lambda - 1$  steps, where each step requires testing to see if each of the  $N_\mu$  points in one column can be connected to each of the  $N_\mu$  points in the next column. Each test must be done, but some of the tests are simpler than others. If the cost at the next grid point is infinite, then there is no point in



doing any further calculations. If on the other hand the cost is finite, then input torque bounds must be checked, and if the input torques are admissible, then costs must be calculated and compared. Though actual computation times will vary with the particular problem being solved, the way the time varies with grid size can be roughly determined. To get a bound on this time, assume that *all* the tests and computations must be performed. Then each step of the dynamic programming algorithm requires  $KN_\mu^2$  seconds, where  $K$  is a quantity which depends upon the computer being used and the number of joints the robot has. There are  $N_\lambda - 1$  such steps, so the time required is less than  $K(N_\lambda - 1)N_\mu^2$ . In other words, the execution time is roughly proportional to the cube of the grid density. In practice, the value of the constant  $K$  must be evaluated experimentally. This has been done for the numerical example in Section 4.3.9, which does indeed show a time dependence proportional to  $(N_\lambda - 1)N_\mu^2$ .

The dependence of execution time on the number of joints  $n$ , i.e., the dependence of the constant  $K$  on  $n$ , is more difficult to assess.  $K$  in the equation above depends on both  $n$  and the representation used to describe the curve to be traversed. The functions  $M_i$  and  $R_i$  depend on the matrices  $J_{ij}$  and  $R_{ij}$  respectively, and the Coriolis term  $Q_i$  depends on the three-dimensional array  $C_{ijk}$ . In general, then, it might be expected that the evaluation of the function  $Q_i$  might take time proportional to the cube of the number of joints. (See, for example, [15].) In any case, the time required for evaluation of the dynamic coefficients is heavily dependent upon the configuration of the robot. Fortunately, in practical cases the number of joints would usually be no more than six, and almost certainly would be less than eight. Since these functions only need to be evaluated once per  $\lambda$ -division of the DP

grid, their evaluation will probably be only a minor part of the total time consumed. This being the case, the dependence of execution time on  $n$  is not an important factor. (For the numerical example considered here, this is certainly true.)

If the algorithm for handling interacting joints is used, then the dependence of the time on the number of joints increases exponentially with the number of joints, since there are  $2^n$  corners on the bounding box for the input. While this would seem to make the algorithm useless, it should be noted that the size of the bounding box decreases as the grid size shrinks. In practice it may be sufficient to test, for example, the endpoints of the current segment, rather than all  $2^n$  corners.

#### 4.3.7. Algorithm Speedup

Even though solution of the trajectory planning problem by dynamic programming requires only a two-dimensional grid, the algorithm uses large amounts of computer time when the grid gets fine enough to give accurate answers. Part of the reason for this is the exhaustive testing of paths in the grid; when connecting points in one column to points in the next, *all* pairs of points are tested. Also, the dynamic programming algorithm generates the optimal trajectories from the *all* points in the grid to the desired goal state. If we can avoid generation of the unused trajectories, considerable speedup should result.

The approach described here involves multiple iterations of the dynamic programming algorithm using a sparse, irregularly spaced grid. Suppose that an approximate solution to the trajectory planning problem is available, say as a result of using dynamic programming with a coarse grid. Then we may plot this approximation in the phase plane, and draw a "swath" around it, indicating the uncertainty of the

solution. Then, instead of superimposing a grid on the entire phase plane, we may superimpose a grid on the uncertainty swath. This grid may have a small, fixed number of  $\mu$  values for each  $\lambda$  value, so that the cost of doing dynamic programming on this grid is relatively low. (Figure 4.3.3)

If the original trajectory met all the constraints, and for each  $\lambda$  value on the grid one of the grid points is on the original trajectory, then a solution will certainly be found when the dynamic programming algorithm is performed. Assuming that the grid includes the points on the upper and lower limits of the swath, the resulting trajectory then must either stay entirely within the swath, or touch the swath's edge. Now a new swath should be drawn, centered on the new optimal trajectory. If the new optimal trajectory touches the edge of the old swath, then the new swath should have the same grid density. If it doesn't touch the edge, then the size of the new swath should be decreased. This process may be repeated until the swath is narrow enough to guarantee that the solution is within desired accuracy limits.

Roughly speaking, the algorithm finds an approximate solution in a reduced search area. If the solution touches the boundary of the search area, then the search area boundary is moved away from the solution. If the solution does not touch the edge of the search area, then a new smaller search area is tried, resulting in a more accurate approximation. By limiting the dynamic programming algorithm's attention to a small area of the phase plane, computation times are kept correspondingly small.

This technique will not be used in any of the examples worked in this section, but a related technique will be described in the Section 4.4.

#### 4.3.8. Convergence Properties

The previous section describes the complexity of the DP algorithm. It is obvious from the discussion that the fineness of the DP grid will have a significant impact on the running time of the algorithm. It will also affect the accuracy of the results. This section describes the effect of the grid density on the accuracy of the DP solution in a quantitative manner.

Bellman proved in [2] that discrete approximations to a continuous optimal control problem will converge (in a sense to be defined) as the step size of the DP stage variable decreases. However, the class of systems to which Bellman's proof applies does not cover those considered in this paper. In particular, Bellman assumes that the dynamic equations of the system are not functions of the stage variable, which is the same as  $\lambda$  in this paper. Here we prove a theorem which is an extension of that of Bellman in that it allows the dynamic equation and cost function to be (possibly discontinuous) functions of the stage variable. The proof presented here also corrects some minor errors in Bellman's proof.

Like Bellman's proof in [2], we will prove that a sequence of discrete dynamic programming processes with decreasing step sizes will produce, under appropriate conditions, a convergent sequence of return functions. It should be noted that the optimal control policy may not converge even though the return functions do. But since the return function is of primary interest, not the details of the control policy, control policy convergence is not generally important.

From the discussion thus far it is clear the the manipulator dynamics and required constraints take the form:

$$\frac{d\mu}{d\lambda} = G(\lambda, \mu, v) \quad (4.3.8.1)$$

where the control variable  $v$  (this is the same as  $\dot{\mu}$  in the previous discussions) must meet some set of constraints:

$$\Omega_q(\lambda, \mu, v) \leq 0, \quad q = 1, 2, \dots, M. \quad (4.3.8.2)$$

Also rewrite the objective function  $J = -C$  to be maximized as follows.

$$J(v) = \Theta(\mu(\lambda_{\max})) + \int_0^{\lambda_{\max}} F(\lambda, \mu, v) d\lambda \quad (4.3.8.3)$$

subject to the initial condition  $\mu(0) = \mu_0$ . Note that the boundary condition  $\mu(\lambda_{\max}) = \mu_f$  can be enforced by taking  $\Theta(\mu(\lambda_{\max}))$  to be zero if  $\mu = \mu_f$  and  $-\infty$  otherwise. The dynamic programming method approximates this continuous problem by discretizing the dynamic equation and objective function using the Euler method, giving:

$$\mu_{k+1} = \mu_k + G(\lambda_k, \mu_k, v_k) \Delta \quad (4.3.8.4)$$

$$J(\{v_k\}) = \Theta(\mu_N) + \sum_{k=0}^{N-1} F(\lambda_k, \mu_k, v_k) \Delta \quad (4.3.8.5)$$

where  $\Delta = \lambda_{\max}/N$ ,  $\lambda_k = k \Delta$ ,  $\mu_k = \mu(\lambda_k)$ ,  $v_k = v(\lambda_k)$ , and the inputs  $v_k$  are constrained by

$$\Omega_q(\lambda_k, \mu_k, v_k) \leq 0, \quad q = 1, 2, \dots, C. \quad (4.3.8.6)$$

Now define  $f_n(c)$  for  $n=0, 1, \dots, N$  by

$$f_n(c) = \text{Sup}_{\{v_k\}} \left[ \Theta(\mu_N) + \sum_{k=N-n}^{N-1} F(\lambda_k, \mu_k, v_k) \Delta \right], \quad \mu_{N-n} = c \quad (4.3.8.7)$$

Then we have

$$f_0(c) = \Theta(c) \quad (4.3.8.8)$$

$$f_{n+1}(c) = \text{Sup}_v \left[ F(\lambda_{N-n-1}, c, v) \Delta + f_n(c + G(\lambda_{N-n-1}, c, v) \Delta) \right] \quad (4.3.8.9)$$

Note that Sup has been used instead of max. This is done to allow the use of non-closed constraint sets and discontinuous functions. It does not materially change the results of the dynamic programming process in that we may make the return function  $f_n$  as close to the optimal value as we please. To see this, consider a single stage of an  $N$ -stage process. For each  $k$  and  $\epsilon > 0$ , we may make  $f_k$  to be within  $\epsilon/2^k$  of its optimal value, thus making  $f_N$  be within  $2\epsilon$  of the optimum. Since  $\epsilon$  may be as small as we please, a control strategy can be constructed which will make the return function agree with the optimum value to within any desired tolerance.

The proof of the main theorem requires the establishment of several lemmas.

**Lemma 4.3.1:** If for the feasible input set  $D$ ,

$$\Gamma(c_1) = \text{Sup}_{v \in D} \left[ \phi_1(c_1, v) + \psi_1(c_1, v) \right]$$

and

$$H(c_2) = \text{Sup}_{v \in D} \left[ \phi_2(c_2, v) + \psi_2(c_2, v) \right]$$

then

$$\left| \Gamma(c_1) - H(c_2) \right| \leq \sup_{v \in D} \left| \phi_1(c_1, v) - \phi_2(c_2, v) \right| + \sup_{v \in D} \left| \psi_1(c_1, v) - \psi_2(c_2, v) \right|.$$

**Proof:** For every  $\epsilon > 0$  there exist  $v_1$  and  $v_2$ , both elements of the set  $D$ , such that

$$\Gamma(c_1) < \phi_1(c_1, v_1) + \psi_1(c_1, v_1) + \epsilon$$

$$H(c_2) < \phi_2(c_2, v_2) + \psi_2(c_2, v_2) + \epsilon$$

We then have the inequalities

$$\phi_1(c_1, v_2) + \psi_1(c_1, v_2) \leq \Gamma(c_1) < \phi_1(c_1, v_1) + \psi_1(c_1, v_1) + \epsilon$$

$$\phi_2(c_2, v_1) + \psi_2(c_2, v_1) \leq H(c_2) < \phi_2(c_2, v_2) + \psi_2(c_2, v_2) + \epsilon$$

Since

$$\Gamma(c_1) - \epsilon \geq \phi_1(c_1, v_2) + \psi_1(c_1, v_2) - \epsilon$$

and

$$H(c_2) - \epsilon < \phi_2(c_2, v_2) + \psi_2(c_2, v_2)$$

we have

$$(\Gamma(c_1) - \epsilon) - (H(c_2) - \epsilon)$$

$$= \Gamma(c_1) - H(c_2)$$

$$> \phi_1(c_1, v_2) - \phi_2(c_2, v_2) + \psi_1(c_1, v_2) - \psi_2(c_2, v_2) - \epsilon$$

Similarly, we have

$$\Gamma(c_1) - H(c_2) < \phi_1(c_1, v_1) - \phi_2(c_2, v_1) + \psi_1(c_1, v_1) - \psi_2(c_2, v_1) + \epsilon$$

But these two conditions imply that

$$\begin{aligned} \left| \Gamma(c_1) - H(c_2) \right| &< \text{Max} \left[ \left| \phi_1(c_1, v_2) - \phi_2(c_2, v_2) + \psi_1(c_1, v_2) - \psi_2(c_2, v_2) - \epsilon \right|, \right. \\ &\quad \left. \left| \phi_1(c_1, v_1) - \phi_2(c_2, v_1) + \psi_1(c_1, v_1) - \psi_2(c_2, v_1) + \epsilon \right| \right] \\ &\leq \text{Max} \left[ \left| \phi_1(c_1, v_2) - \phi_2(c_2, v_2) \right| + \left| \psi_1(c_1, v_2) - \psi_2(c_2, v_2) \right| + \epsilon, \right. \\ &\quad \left. \left| \phi_1(c_1, v_1) - \phi_2(c_2, v_1) \right| + \left| \psi_1(c_1, v_1) - \psi_2(c_2, v_1) \right| + \epsilon \right] \\ &\leq \text{Sup}_{v \in D} \left[ \left| \phi_1(c_1, v) - \phi_2(c_2, v) \right| + \left| \psi_1(c_1, v) - \psi_2(c_2, v) \right| + \epsilon \right] \\ &\leq \text{Sup}_{v \in D} \left| \phi_1(c_1, v) - \phi_2(c_2, v) \right| + \text{Sup}_{v \in D} \left| \psi_1(c_1, v) - \psi_2(c_2, v) \right| + \epsilon \end{aligned}$$

Since  $\epsilon$  may be made as small as we please, the desired result follows. ■

Lemma 4.3.2 shows that the return functions  $f_n$  satisfy a uniform Lipschitz condition which is independent of the step size  $\Delta$ .

**Lemma 4.3.2:** If for all  $c_1, c_2 \in [\mu_{\min}, \mu_{\max}]$  and for all admissible  $\lambda$  and  $v$ , the function  $F$  satisfies a Lipschitz condition

$$\left| F(\lambda, c_1, v) - F(\lambda, c_2, v) \right| \leq K |c_1 - c_2|^\alpha$$



for some  $\alpha > 0$ ,  $\Theta$  satisfies

$$\left| \Theta(c_1) - \Theta(c_2) \right| \leq K |c_1 - c_2|^\alpha$$

and  $G$  satisfies

$$\left| G(\lambda, c_1, v) - G(\lambda, c_2, v) \right| \leq L |c_1 - c_2|^\gamma$$

for some  $\gamma \geq 1$ , then  $f_n$  satisfies the uniform Lipschitz condition

$$\left| f_n(c_1) - f_n(c_2) \right| \leq \Phi |c_1 - c_2|^\alpha \text{ where } \Phi \text{ is independent of } n, c_1, c_2, \text{ and } \Delta,$$

provided that it can be guaranteed that  $\mu_{\min} \leq \mu_k \leq \mu_{\max}$  for all  $0 \leq k \leq N$ .

**Proof:** First we prove by induction that

$$\left| f_n(c_1) - f_n(c_2) \right| \leq k_n \Delta |c_1 - c_2|^\alpha$$

For  $n=0$  we have

$$\left| f_0(c_1) - f_0(c_2) \right| = \left| \Theta(c_1) - \Theta(c_2) \right| \leq K |c_1 - c_2|^\alpha.$$

We may therefore take  $k_0 = \frac{K}{\Delta}$ . Now assume that

$$\left| f_n(c_1) - f_n(c_2) \right| \leq k_n \Delta |c_1 - c_2|^\alpha$$

Then we have

$$\begin{aligned}
& \left| f_{n+1}(c_1) - f_{n+1}(c_2) \right| \\
&= \left| \text{Sup}_{\mathcal{V}} \left[ F(\lambda_{N-n-1}, c_1, v) \Delta + f_n(c_1 + G(\lambda_{N-n-1}, c_1, v) \Delta) \right] \right. \\
&\quad \left. - \text{Sup}_{\mathcal{V}} \left[ F(\lambda_{N-n-1}, c_2, v) \Delta + f_n(c_2 + G(\lambda_{N-n-1}, c_2, v) \Delta) \right] \right|.
\end{aligned}$$

Note that if  $c_1 = \mu_k$ , then  $c_1 + G(\lambda_{N-n-1}, c_1, v) \Delta = \mu_{k+1}$ ; this is where the admissibility of the states  $\mu_k$  for all  $k$  comes into play. Applying Lemma 4.3.1, we have

$$\begin{aligned}
\left| f_{n+1}(c_1) - f_{n+1}(c_2) \right| &\leq \text{Sup}_{\mathcal{V}} \left| F(\lambda_{N-n-1}, c_1, v) - F(\lambda_{N-n-1}, c_2, v) \right| \Delta \\
&\quad + \text{Sup}_{\mathcal{V}} \left| f_n(c_1 + G(\lambda_{N-n-1}, c_1, v) \Delta) \right. \\
&\quad \left. - f_n(c_2 + G(\lambda_{N-n-1}, c_2, v) \Delta) \right|.
\end{aligned}$$

Applying the Lipschitz conditions on  $F$  and  $f_n$ ,

$$\begin{aligned}
\left| f_{n+1}(c_1) - f_{n+1}(c_2) \right| &\leq K \Delta |c_1 - c_2|^\alpha \\
&\quad + k_n \Delta \operatorname{Sup}_v \left| c_1 + G(\lambda_{N-n-1}, c_1, v) \Delta \right. \\
&\quad \left. - c_2 - G(\lambda_{N-n-1}, c_2, v) \Delta \right|^\alpha \\
&= K \Delta |c_1 - c_2|^\alpha \\
&\quad + k_n \Delta \operatorname{Sup}_v \left| (c_1 - c_2) + (G(\lambda_{N-n-1}, c_1, v) \right. \\
&\quad \left. - G(\lambda_{N-n-1}, c_2, v)) \Delta \right|^\alpha \\
&\leq K \Delta |c_1 - c_2|^\alpha + k_n \Delta \left| |c_1 - c_2| \right. \\
&\quad \left. + \operatorname{Sup}_v |G(\lambda_{N-n-1}, c_1, v) - G(\lambda_{N-n-1}, c_2, v)| \Delta \right|^\alpha.
\end{aligned}$$

Applying the Lipschitz condition on  $G$ ,

$$\begin{aligned}
\left| f_{n+1}(c_1) - f_{n+1}(c_2) \right| &\leq K \Delta |c_1 - c_2|^\alpha \\
&\quad + k_n \Delta \left| |c_1 - c_2| + L |c_1 - c_2|^\gamma \Delta \right|^\alpha \\
&= K \Delta |c_1 - c_2|^\alpha \\
&\quad + k_n \Delta \left| |c_1 - c_2| (1 + L |c_1 - c_2|^{\gamma-1} \Delta) \right|^\alpha \\
&\leq K \Delta |c_1 - c_2|^\alpha \\
&\quad + k_n \Delta \left| |c_1 - c_2| (1 + L |\mu_{\max} - \mu_{\min}|^{\gamma-1} \Delta) \right|^\alpha \\
&= K \Delta |c_1 - c_2|^\alpha \\
&\quad + k_n \Delta |c_1 - c_2|^\alpha (1 + L |\mu_{\max} - \mu_{\min}|^{\gamma-1} \Delta)^\alpha.
\end{aligned}$$

Defining  $M \equiv L |\mu_{\max} - \mu_{\min}|^{\gamma-1}$ ,

$$\left| f_{n+1}(c_1) - f_{n+1}(c_2) \right| = K \Delta |c_1 - c_2|^\alpha + k_n \Delta |c_1 - c_2|^\alpha (1 + M \Delta)^\alpha.$$

Using the binomial theorem to expand  $(1 + M \Delta)^\alpha$ ,

$$\begin{aligned} (1 + M \Delta)^\alpha &= 1 + \alpha M \Delta + \frac{\alpha(\alpha-1)}{2!} M^2 \Delta^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!} M^3 \Delta^3 + \dots \\ &= 1 + \alpha M \Delta + \Delta \eta(\Delta). \end{aligned}$$

Since the binomial series converges for  $|M \Delta| < 1$ , the function  $\eta(\Delta)$  is bounded, and goes to zero as  $\Delta$  goes to zero. Therefore, for  $\Delta$  sufficiently small, we may take

$$(1 + M \Delta)^\alpha \leq 1 + (\alpha M + \epsilon) \Delta = 1 + P \Delta$$

where  $\epsilon$  is some small positive constant. This yields the inequality

$$\left| f_{n+1}(c_1) - f_{n+1}(c_2) \right| \leq K \Delta |c_1 - c_2|^\alpha + k_n \Delta |c_1 - c_2|^\alpha (1 + P \Delta).$$

We may then take  $k_{n+1}$  to be defined by the recurrence relation

$$k_{n+1} = K + k_n (1 + P \Delta).$$

The solution of this equation which satisfies the initial condition  $k_0 = K$  is

$$k_n = \frac{K}{P \Delta} \left[ (1 + P \Delta)^{n+1} - 1 \right].$$

Then at every stage  $n$  we have

$$\left| f_n(c_1) - f_n(c_2) \right| \leq k_n \Delta |c_1 - c_2|^\alpha = \frac{K}{P} \left[ (1 + P \Delta)^{n+1} - 1 \right] |c_1 - c_2|^\alpha.$$

Since  $k_n \leq k_N$  for  $n \leq N$ , we may use the condition

$$\left| f_n(c_1) - f_n(c_2) \right| \leq k_N \Delta |c_1 - c_2|^\alpha$$

for all  $n \leq N$ . Now since  $\Delta = \lambda_{\max}/N$ , we have

$$\left| f_n(c_1) - f_n(c_2) \right| \leq \frac{K}{P} \left[ (1 + P\Delta)^{1 + \frac{\lambda_{\max}}{\Delta}} - 1 \right] |c_1 - c_2|^\alpha.$$

As  $\Delta \rightarrow 0$ , the quantity in brackets remains finite, and in fact approaches the limit  $e^{P\lambda_{\max}} - 1$ . Since this quantity is also finite for all  $\Delta > 0$ , we may construct the inequality

$$\left| f_n(c_1) - f_n(c_2) \right| \leq \Phi |c_1 - c_2|^\alpha$$

where the constant  $\Phi$  is given by

$$\Phi = \frac{K}{P} \max_{0 < \Delta \leq \lambda_{\max}} \left[ (1 + P\Delta)^{1 + \frac{\lambda_{\max}}{\Delta}} - 1 \right].$$

Since  $\Phi$  is independent of  $n$ ,  $c_1$ ,  $c_2$ , and  $\Delta$ , we have the uniform Lipschitz condition desired. ■

Lemma 4.3.3 establishes a connection between the dynamic programming process with step size  $\Delta$  and that with step size  $2\Delta$ .

**Lemma 4.3.3:** Let the process  $g_k$  with step size  $2\Delta$  satisfy the recurrence relations

$$g_0(c) = \Theta(c)$$

$$g_{k+1}(c) = \text{Sup}_v \left[ F(\lambda_{N-2k-2}, c, v)2\Delta + g_k(c + G(\lambda_{N-2k-2}, c, v)2\Delta) \right].$$

Define the auxiliary process  $h_{2k}$  with step size  $2\Delta$  by the equations

$$h_0(c) = \Theta(c)$$

$$\begin{aligned} h_{2k+2}(c) = \text{Sup}_v \left[ F(\lambda_{N-2k-2}, c, v)\Delta + F(\lambda_{N-2k-1}, c + G(\lambda_{N-2k-2}, c, v)\Delta, v)\Delta \right. \\ \left. + h_{2k}(c + G(\lambda_{N-2k-2}, c, v)\Delta \right. \\ \left. + G(\lambda_{N-2k-1}, c + G(\lambda_{N-2k-2}, c, v)\Delta, v)\Delta) \right]. \end{aligned}$$

This process can be thought of as a process with step size  $\Delta$  in which the input policy is restricted so that the input for the  $N-2k-2^{\text{nd}}$  interval is the same as for the  $N-2k-1^{\text{st}}$  interval. Let  $F, G$  and  $\Theta$  satisfy the Lipschitz conditions

$$|F(\lambda_1, c_1, v) - F(\lambda_2, c_2, v)| \leq K |c_1 - c_2|^\alpha + B |\lambda_1 - \lambda_2|^\beta$$

$$|F(\lambda, c_1, v) - F(\lambda, c_2, v)| \leq K |c_1 - c_2|^\alpha$$

$$|\Theta(c_1) - \Theta(c_2)| \leq K |c_1 - c_2|^\alpha$$

$$|G(\lambda_1, c_1, v) - G(\lambda_2, c_2, v)| \leq L |c_1 - c_2|^\gamma + C |\lambda_1 - \lambda_2|^\delta$$

$$|G(\lambda, c_1, v) - G(\lambda, c_2, v)| \leq L |c_1 - c_2|^\gamma$$

where  $K, B, L$ , and  $C$  are constants;  $\alpha > 0$ ,  $\gamma \geq 1$ ,  $\beta \geq 0$ , and  $\delta \geq 0$ , for all admissible  $\lambda, c_1, c_2$ , and  $v$ , and for all  $\lambda_1$ , and  $\lambda_2$  such that the interval  $[\min(\lambda_1, \lambda_2), \max(\lambda_1, \lambda_2)]$  does not contain any of the  $N_d$  points of discontinuity

$d_1, d_2, \dots, d_{N_i}$ . Also let  $F$  and  $G$  satisfy

$$|F(\lambda_1, c_1, v) - F(\lambda_2, c_2, v)| \leq K |c_1 - c_2|^\alpha + B$$

and

$$|G(\lambda_1, c_1, v) - G(\lambda_2, c_2, v)| \leq L |c_1 - c_2|^\gamma + C$$

for all admissible  $\lambda_1, \lambda_2, c_1, c_2$ , and  $v$ . Then

$$|h_{2k}(c_1) - g_k(c_2)| \leq \Phi |c_1 - c_2|^\alpha + E \Delta^\omega$$

where  $\omega = \min\{1, \alpha, \beta, \alpha(1+\gamma)-1, \alpha(1+\delta)-1\}$ , and  $\Phi$  and  $E$  are constants.

**Proof:** We first prove that there exist numbers  $p_k$  such that

$$|h_{2k}(c_1) - g_k(c_2)| \leq \Phi |c_1 - c_2|^\alpha + p_k \Delta^{\omega+1}.$$

We proceed by induction. We have

$$|h_0(c_1) - g_0(c_2)| = |\Theta(c_1) - \Theta(c_2)| \leq K |c_1 - c_2|^\alpha$$

so the result holds for  $k = 0$  with  $p_0 = 0$ . Now assume that

$$|h_{2k}(c_1) - g_k(c_2)| \leq \Phi |c_1 - c_2|^\alpha + p_k \Delta^{\omega+1}$$

for some particular value of  $k$ . Applying Lemma 4.3.2 to the function  $g_{k+1}$ , we have

$$\begin{aligned} & |h_{2k+2}(c_1) - g_{k+1}(c_2)| \\ &= |h_{2k+2}(c_1) - g_{k+1}(c_1) + g_{k+1}(c_1) - g_{k+1}(c_2)| \\ &\leq |h_{2k+2}(c_1) - g_{k+1}(c_1)| + |g_{k+1}(c_1) - g_{k+1}(c_2)| \end{aligned}$$

$$\begin{aligned}
&\leq \Phi |c_1 - c_2|^\alpha + |h_{2k+2}(c_1) - g_{k+1}(c_1)| \\
&= \Phi |c_1 - c_2|^\alpha \\
&\quad + \left| \text{Sup}_v \left[ F(\lambda_{N-2k-2}, c_1, v) \Delta \right. \right. \\
&\qquad\qquad\qquad + F(\lambda_{N-2k-1}, c_1 + G(\lambda_{N-2k-2}, c_1, v) \Delta, v) \Delta \\
&\qquad\qquad\qquad + h_{2k}(c_1 + G(\lambda_{N-2k-2}, c_1, v) \Delta) \\
&\qquad\qquad\qquad \left. \left. + G(\lambda_{N-2k-1}, c_1 + G(\lambda_{N-2k-2}, c_1, v) \Delta, v) \Delta \right] \right. \\
&\quad \left. - \text{Sup}_v \left[ F(\lambda_{N-2k-2}, c_1, v) 2\Delta \right. \right. \\
&\qquad\qquad\qquad \left. \left. + g_k(c_1 + G(\lambda_{N-2k-2}, c_1, v) 2\Delta) \right] \right|.
\end{aligned}$$

Applying Lemma 4.3.1,

$$\begin{aligned}
&|h_{2k+2}(c_1) - g_{k+1}(c_1)| \\
&\leq \text{Sup}_v \left| F(\lambda_{N-2k-1}, c_1 + G(\lambda_{N-2k-2}, c_1, v) \Delta, v) - F(\lambda_{N-2k-2}, c_1, v) \right| \Delta \\
&\quad + \text{Sup}_v \left| h_{2k}(c_1 + G(\lambda_{N-2k-2}, c_1, v) \Delta) \right. \\
&\quad\quad\quad \left. + G(\lambda_{N-2k-1}, c_1 + G(\lambda_{N-2k-2}, c_1, v) \Delta, v) \Delta \right|
\end{aligned}$$



$$\begin{aligned}
& -g_k(c_1 + G(\lambda_{N-2k-2}, c_1, v)2\Delta) \Big| \\
& \leq \text{Sup}_v K |G(\lambda_{N-2k-2}, c_1, v)\Delta|^\alpha \Delta + B\Delta^{1+\beta} \\
& + \Phi \text{Sup}_v \left| G(\lambda_{N-2k-1}, c_1 + G(\lambda_{N-2k-2}, c_1, v)\Delta, v)\Delta \right. \\
& \qquad \qquad \qquad \left. - G(\lambda_{N-2k-2}, c_1, v)\Delta \right|^\alpha + p_k \Delta^{\omega+1}
\end{aligned}$$

if the interval  $[\lambda_{N-2k-2}, \lambda_{N-2k}]$  does not contain any of the points  $d_1, \dots, d_{N_i}$ .

Applying the Lipschitz condition on  $G$ , we have

$$\begin{aligned}
& |h_{2k+2}(c_1) - g_{k+1}(c_1)| \\
& \leq \text{Sup}_v K |G(\lambda_{N-2k-2}, c_1, v)\Delta|^\alpha \Delta + B\Delta^{1+\beta} \\
& \qquad \qquad \qquad + \Phi \text{Sup}_v \left| L |G(\lambda_{N-2k-2}, c_1, v)\Delta|^\gamma + C\Delta^\delta \right|^\alpha \Delta^\alpha + p_k \Delta^{\omega+1}
\end{aligned}$$

or

$$\begin{aligned}
| h_{2k+2}(c_1) - g_{k+1}(c_1) | & \leq Z \Delta^{\alpha+1} + B \Delta^{1+\beta} + A \Delta^{\alpha(1+\eta)} + p_k \Delta^{\omega+1} \\
& = (Z \Delta^{\alpha-\omega} + B \Delta^{\beta-\omega} + A \Delta^{\alpha(1+\eta)-1-\omega} + p_k) \Delta^{\omega+1}
\end{aligned}$$

where  $\eta = \min(\gamma, \delta)$ ,  $A = \Phi \text{Sup}_v \left| L |G(\lambda_{N-2k-2}, c_1, v)|^\gamma \Delta^{\gamma-\eta} + C \Delta^{\delta-\eta} \right|^\alpha$ , and

$$Z = \text{Sup}_v K |G(\lambda_{N-2k-2}, c_1, v)|^\alpha.$$

Now  $\omega$  is defined by  $\omega = \min\{1, \alpha, \beta, \alpha(1+\gamma)-1, \alpha(1+\delta)-1\}$ , so  $\omega$  is less than or equal to  $\alpha$ ,  $\beta$ , and  $\alpha(1+\eta)-1$ . Since we also have  $\Delta \leq \lambda_{\max}$ , we have

$$|h_{2k+2}(c_1) - g_{k+1}(c_1)| \leq (D + p_k) \Delta^{\omega+1}$$

for the constant  $D = Z \Delta^{\alpha-\omega} + B \Delta^{\beta-\omega} + A \Delta^{\alpha(1+\eta)-1-\omega}$ . This gives

$$|h_{2k+2}(c_1) - g_{k+1}(c_2)| \leq \Phi |c_1 - c_2|^\alpha + (D + p_k) \Delta^{\omega+1}$$

If, on the other hand, the interval  $[\lambda_{N-2k-2}, \lambda_{N-2k}]$  contains one of the  $d_i$ , then we have

$$\begin{aligned} |h_{2k+2}(c_1) - g_{k+1}(c_1)| &\leq \underset{v}{\text{Sup}} K |G(\lambda_{N-2k-2}, c_1, v) \Delta|^\alpha \Delta + B \Delta \\ &\quad + \Phi \underset{v}{\text{Sup}} |L |G(\lambda_{N-2k-2}, c_1, v) \Delta|^\gamma + C|^\alpha \Delta^\alpha + p_k \Delta^{\omega+1} \end{aligned}$$

or

$$|h_{2k+2}(c_1) - g_{k+1}(c_1)| \leq W \Delta^\zeta + p_k \Delta^{\omega+1}$$

where  $\zeta = \min(1, 1+\alpha, \alpha) = \min(1, \alpha)$ , and

$$\begin{aligned} W &= \underset{v}{\text{Sup}} K |G(\lambda_{N-2k-2}, c_1, v)|^\alpha \Delta^{1+\alpha-\zeta} + B \Delta^{1-\zeta} \\ &\quad + \Phi \underset{v}{\text{Sup}} |L |G(\lambda_{N-2k-2}, c_1, v) \Delta|^\gamma|^\alpha \Delta^{\alpha-\zeta}. \end{aligned}$$

We may therefore take

$$p_{k+1} = p_k + \begin{cases} W \Delta^{\zeta-\omega-1} & \text{if some } d_i \in [\lambda_{N-2k-2}, \lambda_{N-2k}] \\ D & \text{otherwise} \end{cases}$$

But, then we have  $p_k \leq p_N = p_0 + (N - N_d)D + N_d W \Delta^{\zeta-\omega-1}$  for all  $k \leq N$ .

Noting that  $\lambda_{\max} = 2N \Delta$ ,

$$\begin{aligned}
|h_{2k}(c_1) - g_k(c_2)| &\leq \Phi |c_1 - c_2|^\alpha + ((N - N_d)D + N_d W \Delta^{\zeta-\omega-1} + p_0) \Delta^{\omega+1} \\
&= \Phi |c_1 - c_2|^\alpha + \frac{\lambda_{\max} D}{2} \Delta^\omega - N_d D \Delta^{\omega+1} + N_d W \Delta^\zeta \\
&\leq \Phi |c_1 - c_2|^\alpha + E \Delta^\omega
\end{aligned}$$

since, again,  $\omega \leq \min(1, \alpha) = \zeta$ . ■

Lemma 4.3.4 relates to convergence of infinite sequences.

**Lemma 4.3.4:** If  $\infty > W \geq u_{n+1} \geq u_n - a_n$ , where  $a_n \geq 0$  and  $\sum a_n < \infty$ , then the sequence  $\{u_n\}$  converges.

**Proof:** Let  $x_M$  and  $x_P$  be two cluster points of the sequence  $\{u_n\}$ . Let the sequences  $\{u_{M_i}\}$  and  $\{u_{P_i}\}$  converge to  $x_M$  and  $x_P$  respectively, and let

$$M_1 < P_1 < M_2 < P_2 < M_3 < P_3 < \dots$$

Then we have

$$u_{M_i} - u_{P_i} \geq - \sum_{k=M_i}^{P_i} a_k = -\epsilon_i$$

and

$$u_{P_i} - u_{M_{i+1}} \geq - \sum_{k=P_i}^{M_{i+1}} a_k = -\delta_i$$

Since the series  $\sum a_n$  converges,  $\epsilon_i$  and  $\delta_i$  go to zero as  $i \rightarrow \infty$ . But the left sides of these inequalities approach  $x_M - x_P$  and  $x_P - x_M$  respectively. Therefore, we have  $x_M - x_P \geq 0$  and  $x_P - x_M \geq 0$ , or  $x_M = x_P$ . ■

We are now in a position to state and prove the main theorem. It shows that the return functions for the dynamic programming problem converge, provided that the functions  $F$  and  $G$  satisfy some Lipschitz conditions everywhere except at a finite number of jump discontinuities, and provided that the variable  $\mu$  can be guaranteed to stay within appropriate bounds.

**Theorem 4.3.1:** Let the input  $v$  satisfy  $0 \leq v \leq 1$ , and let  $F$  and  $G$  satisfy the Lipschitz conditions

$$|F(\lambda_1, c_1, v) - F(\lambda_2, c_2, v)| \leq K |c_1 - c_2|^\alpha + B |\lambda_1 - \lambda_2|^\beta$$

$$|F(\lambda, c_1, v) - F(\lambda, c_2, v)| \leq K |c_1 - c_2|^\alpha$$

$$|G(\lambda_1, c_1, v) - G(\lambda_2, c_2, v)| \leq L |c_1 - c_2|^\gamma + C |\lambda_1 - \lambda_2|^\delta$$

$$|G(\lambda, c_1, v) - G(\lambda, c_2, v)| \leq L |c_1 - c_2|^\gamma$$

where  $\alpha > 0$ ,  $\gamma \geq 1$ ,  $\beta \geq 0$ , and  $\delta \geq 0$ , for all admissible  $\lambda$  and  $v$ , for all  $\mu_{\min} \leq c_1, c_2 \leq \mu_{\max}$ , and for all  $\lambda_1$ , and  $\lambda_2$  such that the interval  $[\min(\lambda_1, \lambda_2), \max(\lambda_1, \lambda_2)]$  does not contain any of the  $N_d$  points of discontinuity  $d_1, d_2, \dots, d_{N_d}$ . Also let  $F$  and  $G$  satisfy

$$|F(\lambda_1, c_1, v) - F(\lambda_2, c_2, v)| \leq K |c_1 - c_2|^\alpha + B$$

and

$$|G(\lambda_1, c_1, v) - G(\lambda_2, c_2, v)| \leq L |c_1 - c_2|^\gamma + C$$

for all admissible  $\lambda_1$ ,  $\lambda_2$ ,  $c_1$ ,  $c_2$ , and  $v$ . Then the discrete dynamic programming pro-

cess yields return functions which converge to a limit as the step size  $\Delta$  goes to zero, provided that  $\mu_k$  can be guaranteed to stay in the interval  $[\mu_{\min}, \mu_{\max}]$ .

**Proof:** Consideration of the intermediate process  $h_{2k}$  described in Lemma 4.3.3 shows that  $f_{2k}(c) \geq h_{2k}(c)$ , so by Lemma 4.3.3 we may write

$$f_{2k}(c) \geq h_{2k}(c) \geq g_k(c) - E \Delta^\omega$$

for some  $\omega > 0$ . If  $F$  is to satisfy the required Lipschitz conditions, then  $F$  must be bounded, so that  $f$  remains finite. Therefore if we define  $u_r$  to be the return at stage  $2^r k$  with step size  $\Delta/2^r$ , then we have

$$\infty > W \geq u_{r+1} \geq u_r - E \left( \frac{\Delta}{2^r} \right)^\omega$$

Therefore by Lemma 4.3.4 the sequence  $\{u_r\}$  converges. ■

Roughly speaking, this theorem shows that when the functions  $F$  and  $G$  are continuous of sufficiently high order in  $\mu$  and piecewise continuous in the stage variable  $\lambda$ , then the discrete dynamic programming process converges.

One important point is that in order to apply the theorem, it must be guaranteed that  $\mu_k$  stays within the range in which the Lipschitz conditions are valid. This can be accomplished in several ways. If, for example,  $G(\lambda, \mu_{\max}, v) \leq 0$  and  $G(\lambda, \mu_{\min}, v) \geq 0$  for all  $\lambda$ , then the optimal trajectory can never escape the interval  $[\mu_{\min}, \mu_{\max}]$ . Another way to assure containment in this interval is to construct an objective function which guarantees that trajectories which stray outside the interval are heavily penalized and therefore never selected. This method works

for the examples in this paper; since the examples all penalize time, they all have terms which are inversely proportional to velocity, and so keep the velocity  $\mu$  greater than some  $\epsilon > 0$ .

#### 4.3.9. Numerical Examples

To demonstrate the use of the dynamic programming algorithm, we present several examples. These examples use the first three joints of a cylindrical electrically-driven manipulator, the Bendix PACS arm. The second and third joints of this robot are similar to the two degree-of-freedom robot used to demonstrate the phase plane method, so a direct comparison of the methods is possible in those cases in which the cost function is minimum time. In addition, several examples will treat cases to which the phase plane technique does not apply.

Before presenting the example, an explicit form of the objective function must be chosen. The objective function used here has one component proportional to traversal time,  $T$ , and another component proportional to frictional and electrical energy losses. The servo drives of the arm are assumed to consist of a voltage source in series with a resistor and an ideal DC motor. This gives the form

$$C = r_t T + r_e \int_0^T \mathbf{R}_{ij} \dot{\mathbf{q}}^i \dot{\mathbf{q}}^j dt + r_e \int_0^T \sum_i I_i^2 R_i^m dt \quad (4.3.9.1)$$

$$= r_t \int_0^{\lambda^*} \frac{1}{\mu} d\lambda + r_e \int_0^{\lambda^*} \mu^2 \mathbf{R}_{ij} \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} d\lambda \quad (4.3.9.2)$$

$$+ r_e \int_0^T \frac{1}{\mu} \sum_i I_i^2 R_i^m d\lambda$$

where  $r_t$  and  $r_e$  are related to revenue generated per item and the energy costs of

the motion respectively, and  $I_i$  and  $R_i^m$  are the motor currents and resistances for joint  $i$ . Since the joint torques  $u_i$  are related to the motor currents  $I_i$  by the relationships

$$u_i = \frac{k_i^m}{k_i^g} I_i \quad (4.3.9.3)$$

where  $k_i^m$  and  $k_i^g$  are the motor constant and the motor gearing respectively, the sum in Eq. (4.3.9.2) can be written as  $\sum_i \frac{(k_i^g)^2 R_i^m}{(k_i^m)^2} u_i^2$ . The torques  $u_i$  are given in terms of  $\lambda$ ,  $\mu$ , and  $\dot{\mu}$  by Eq. (4.1.6), which is quadratic in  $\mu$ , so the integral in Eq. (4.3.9.2) can be expressed in terms of integrals of powers of  $\mu$ .

Since the path is known over one  $\lambda$ -interval, we can determine the components of the incremental cost. To do this, we need the integrals  $\int_{\lambda_t}^{\lambda_{t+1}} \frac{1}{\mu} d\lambda$ ,  $\int_{\lambda_t}^{\lambda_{t+1}} \mu d\lambda$ ,  $\int_{\lambda_t}^{\lambda_{t+1}} \mu^2 d\lambda$ ,  $\int_{\lambda_t}^{\lambda_{t+1}} \mu^3 d\lambda$ . In general, we have

$$\begin{aligned} \Phi_k &\equiv \int_{\lambda_t}^{\lambda_{t+1}} \mu^n d\lambda = \int_{\lambda_t}^{\lambda_{t+1}} \left[ \frac{(\lambda_{k+1} - \lambda)\mu_0^2 + (\lambda - \lambda_k)\mu_1^2}{\lambda_{k+1} - \lambda_k} \right]^{\frac{n}{2}} d\lambda \\ &= \frac{\lambda_{k+1} - \lambda_k}{(1 + \frac{n}{2})(\mu_1^2 - \mu_0^2)} (\mu_1^{n+2} - \mu_0^{n+2}). \end{aligned} \quad (4.3.9.4)$$

Equations (4.3.9.2), (4.3.9.4), and (4.1.1) give the incremental cost as

$$\begin{aligned}
C = & 2r_t \Phi_{-1} + r_e \mathbf{R}_{ij} \frac{df^i}{d\lambda} \frac{df^j}{d\lambda} \Phi_1 & (4.3.9.5) \\
& + r_e \sum_i \frac{(k_i^j)^2 R_i^m}{(k_i^m)^2} \left[ Q_i^2 \Phi_3 + 2Q_i R_i \Phi_2 + (R_i^2 + 2Q_i (S_i + M_i \dot{\mu})) \Phi_1 \right. \\
& \left. + 2R_i (S_i + M_i \dot{\mu}) \Phi_0 + (S_i + M_i \dot{\mu})^2 \Phi_{-1} \right]
\end{aligned}$$

It should be noted that this cost function always has a  $\Phi_{-1}$  term unless the time penalty is zero and the robot is not influenced by gravity. (In this case, the optimal solution is not to move at all!) Therefore the cost function will prevent the trajectory from going to zero velocity unless forced by boundary conditions. Also, since torques and motor voltages are bounded, the velocity  $\mu$  is bounded. Thus  $\mu$  in the DP algorithm stays within some interval  $[\mu_{\min}, \mu_{\max}]$ , as is required for convergence. Since the maximum and minimum values of the control variable  $\dot{\mu}$  can be computed from Eq. (4.1.1), we may define a new control variable  $\rho$  by the relationship

$$\dot{\mu} = \dot{\mu}_{\min}(\lambda, \mu) + (\dot{\mu}_{\max}(\lambda, \mu) - \dot{\mu}_{\min}(\lambda, \mu))\rho \quad (4.3.9.6)$$

so that  $\rho$  ranges from zero to one. It is easily shown that the other conditions of the convergence theorem are met if the parameterized curve is suitably well behaved, so the DP algorithm will converge with this cost function.

The dynamic equations and actuator characteristics for the PACS arm are given in the Appendix. The DP algorithm was implemented in the C programming language running under UNIX on a VAX11-780. The parameterized curves are represented as sequences of points. All computations are done for a path which is a



straight line from (0.7,0.7,0.1) to (0.4,-0.4,0.4), all (Cartesian) coordinates being given in meters.

To verify the correctness of the dynamic programming algorithm, it was first applied to the simple two-degree-of-freedom robot which was used in section 4.2.5. The phase plane plots for minimum time, with a  $10 \times 10$  and a  $40 \times 40$  DP grid are plotted in Figures 4.3.4 and 4.3.5, along with the phase plane plots calculated by the phase plane method. Reassuringly, the trajectories calculated by the DP method seem to converge to the correct minimum-time phase plane plot as the grid gets finer.

Figure 4.3.6a shows the phase plane plot for a  $10 \times 10$  grid with a pure minimum time cost function. Figures 4.3.6b through 4.3.6d show joint torque or force versus time, and Figures 4.3.6e shows motor voltage vs. time. Figures 4.3.7 are the same, except that the grid is 40 columns by 160 rows. The calculated traversal times for  $10 \times 10$ ,  $20 \times 40$ , and  $40 \times 160$  grids are 2.000, 1.972, and 1.905 seconds respectively, compared to 1.782 seconds as calculated by the phase plane method.

Figures 4.3.8 and 4.3.9 show phase plane, torque, and motor voltage plots for a time penalty of 1 unit per second and an energy penalty of ten units per joule. Note that the trajectory is lower than that which is obtained if only time is penalized. The grid sizes are  $10 \times 10$  and  $20 \times 40$ .

Figures 4.3.10 and 4.3.11 show the results for a minimum time trajectory when, in addition to the torque and voltage constraints given in Table A.2, the total power sunk or sourced by the robot is limited to 2 kilowatts.

The time consumed by the algorithm was measured for several different grid sizes, with, however, 400 interpolation points on the curve regardless of grid size.

Computation of the dynamic coefficients for 400 points usually took from 0.350 to 1.0 seconds of real time, so the computation time is probably about 0.35 seconds. The computation times for the dynamic coefficients vary as the cube of the number of joints. Thus for a robot with six degrees of freedom instead of three we would expect about eight times as much computation. Taking 100 times the 0.35 seconds, to be very conservative, gives 35 seconds for 400 points. But 400 interpolation points are hardly necessary for a grid with a value of 40 for  $N_\lambda$ ; 80 points would certainly be adequate, giving a computation time of about 7 seconds for the dynamic coefficients, not an unreasonable figure if the motion is to be repeated a large number of times.

The times given in Table 4.3.1 are real times for the DP algorithm on a lightly loaded system (average of approximately 2 tasks running concurrently). The times must therefore be regarded as approximations to the actual computation times. Table 4.3.1 also lists the function  $6 \times 10^{-4} N_\lambda N_\mu^2$ , which seems to give a good match to the actual running time, as predicted in Section 4.3.6. It should be noted that the program was run with all debugging features enabled, and that no serious attempt was made to optimize the source code. Indeed, there are redundant computations in several places which could be eliminated.

The effect of grid size on the quality of the results of the dynamic programming algorithm is of practical importance. The grid must be fine enough to give good results but not so fine that the time required to perform the DP algorithm is excessive. Intuitively, varying the number of rows and number of columns in the grid will have different effects on the results. Varying the number of columns (the number of  $\lambda$ -divisions) varies the accuracy of the dynamic model; using a smaller size yields a

more accurate approximation to the true dynamic model, since the "pieces" in the piecewise-constant dynamic model will be smaller. Varying the number of rows (the number of  $\mu$ -divisions) varies the accuracy of the approximation to the true minimum-cost solution; a finer grid size will in general yield a better approximation.

Another important factor is the ratio of the number of rows to the number of columns. If the number of columns is very large and the number of rows is very small, then the slope of the curves connecting one point in the DP grid to another must be either zero or very large. Since the torque bounds induce bounds on the slope of the curve, it is possible that the DP algorithm may not even be able to connect a point to its nearest diagonal neighbor. In this case, the algorithm will give no solution at all. Therefore when choosing grid size, one must (i) choose the  $\lambda$ -divisions to be small enough to make the piecewise-constant dynamic model of the robot sufficiently accurate, (ii) choose the  $\mu$ -divisions to be small enough so that the resulting trajectory is a satisfactory approximation to the true minimum-cost trajectory, and (iii) make sure that the  $\mu$ -divisions are small enough so that the slope of a curve connecting two adjacent points in the grid is small.

$N_\lambda$	$N_\mu$	minimum	average	predicted
10	10	1.217	1.355	0.6
10	20	3.583	5.260	2.4
20	40	19.917	25.5	19.2
20	80	92.050	110.367	76.8
40	40	29.967	33.856	38.4

Table 4.3.1. Computation times for different grid sizes (seconds)

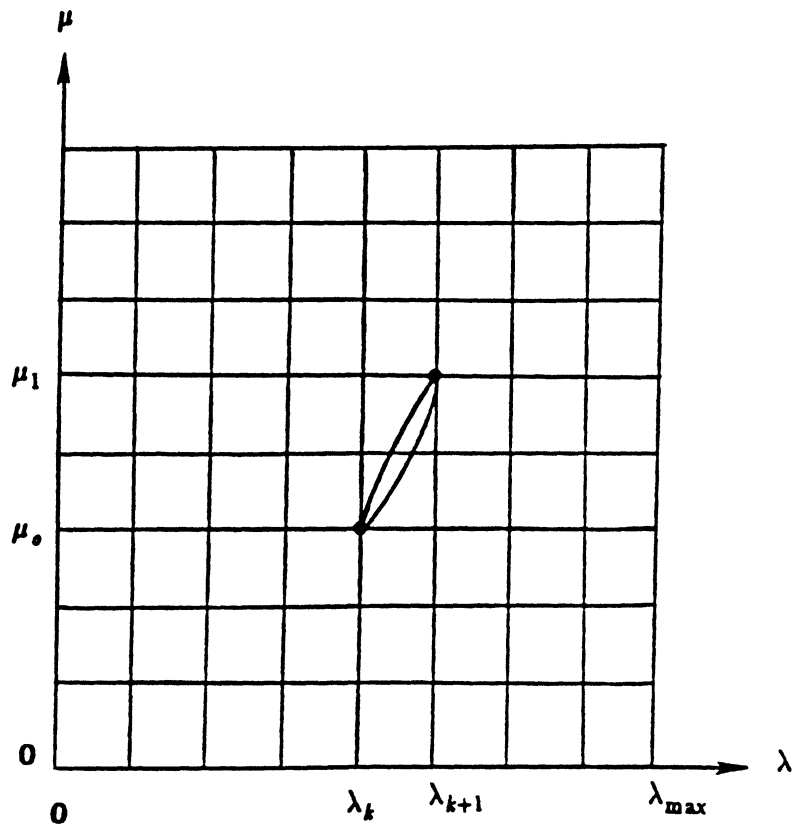


Figure 4.3.1. Curves connecting adjacent DP grid points

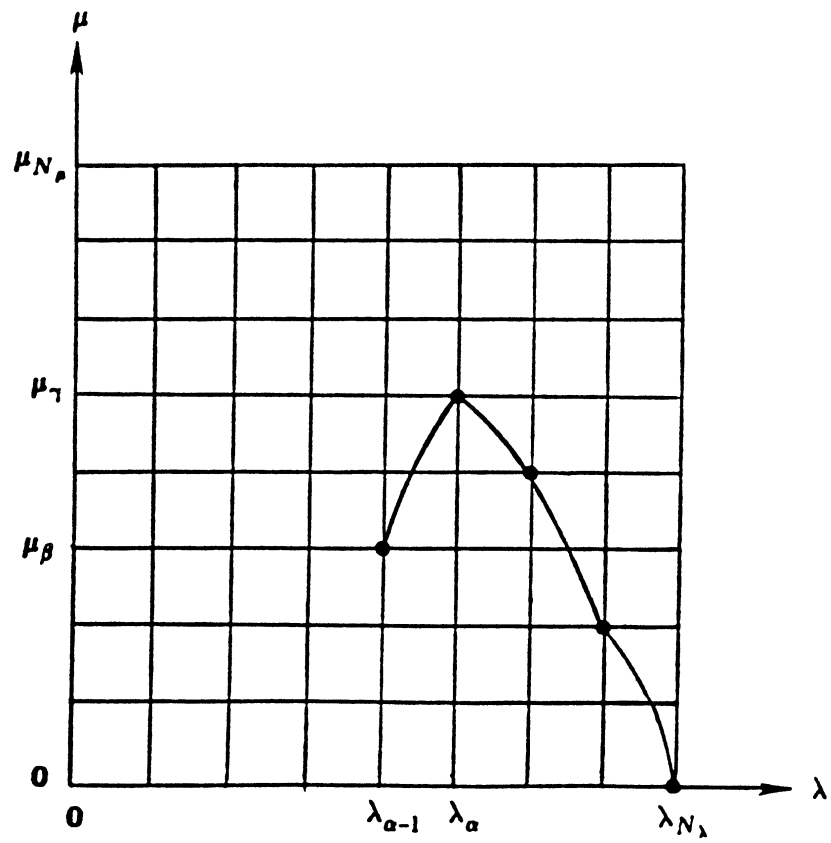


Figure 4.3.2. Curve connecting a grid point to an existing trajectory

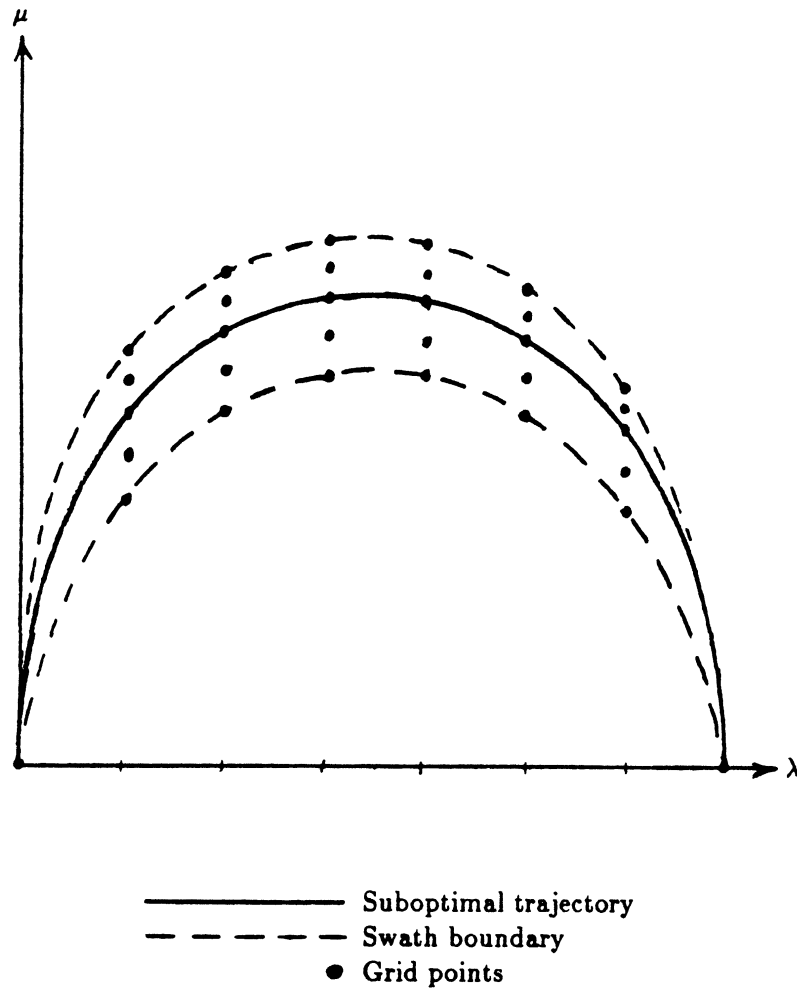


Figure 4.3.3. Suboptimal trajectory with uncertainty swath

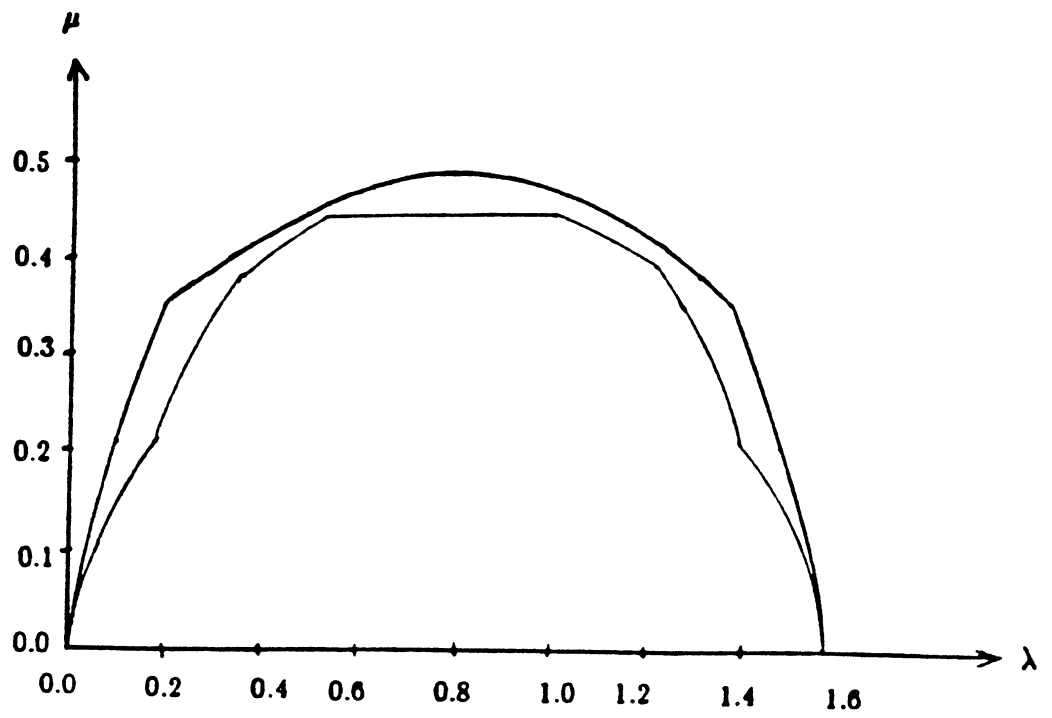


Figure 4.3.4. Minimum time phase plane plot for polar manipulator, 10 $\times$ 10 grid



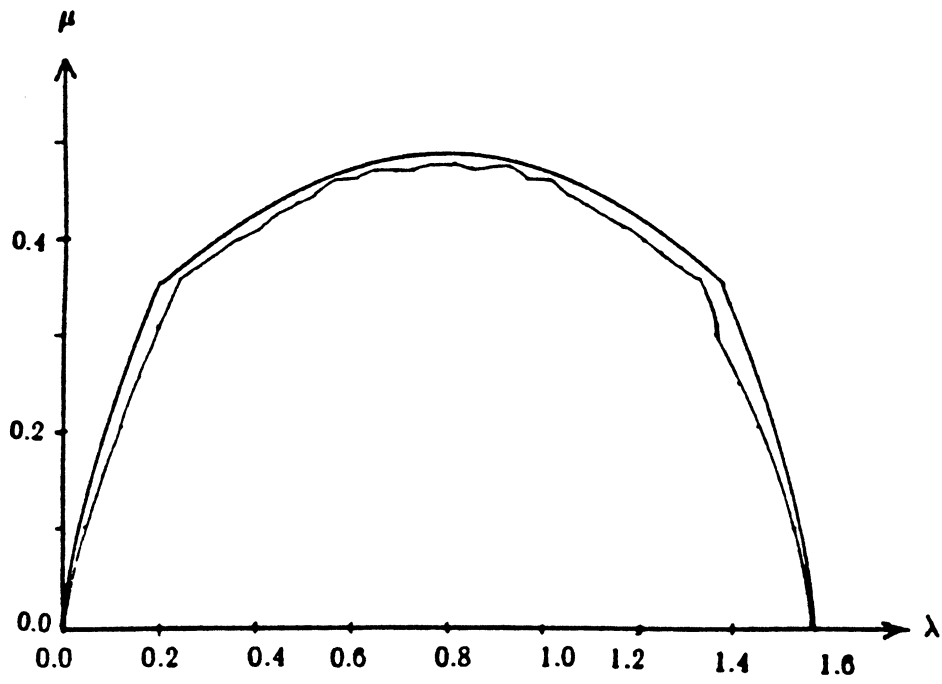


Figure 4.3.5. Minimum time phase plane plot for polar manipulator, 40 $\times$ 40 grid

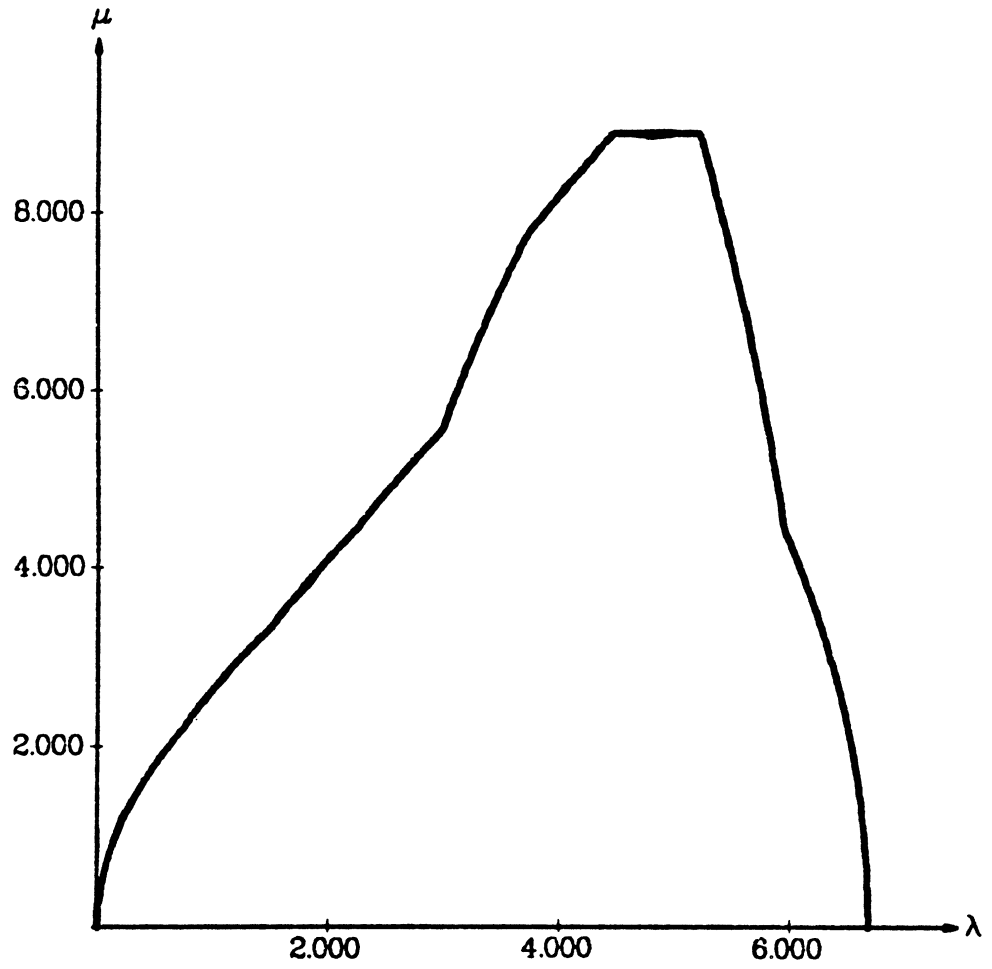


Figure 4.3.6a. Minimum time phase plot for PACS arm, 10 $\times$ 10 grid

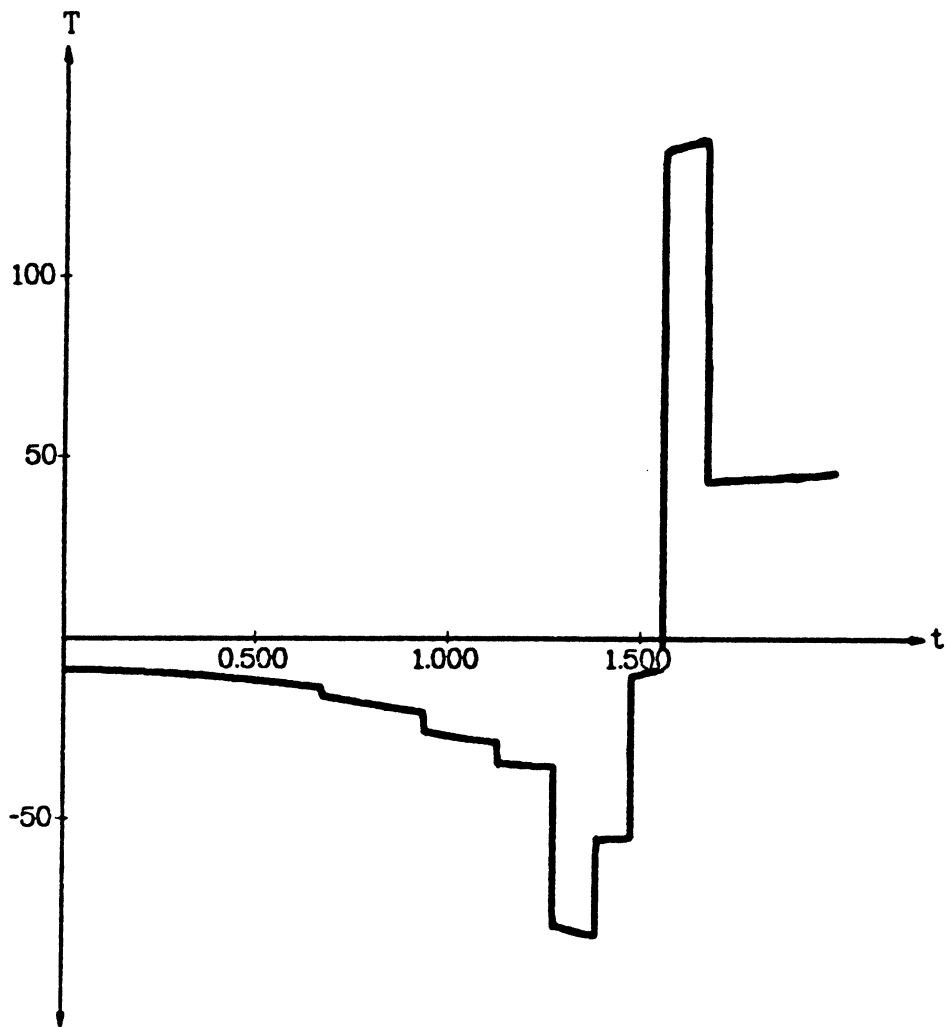


Figure 4.3.6b.  $\theta$  joint torque vs. time,  $10 \times 10$  grid

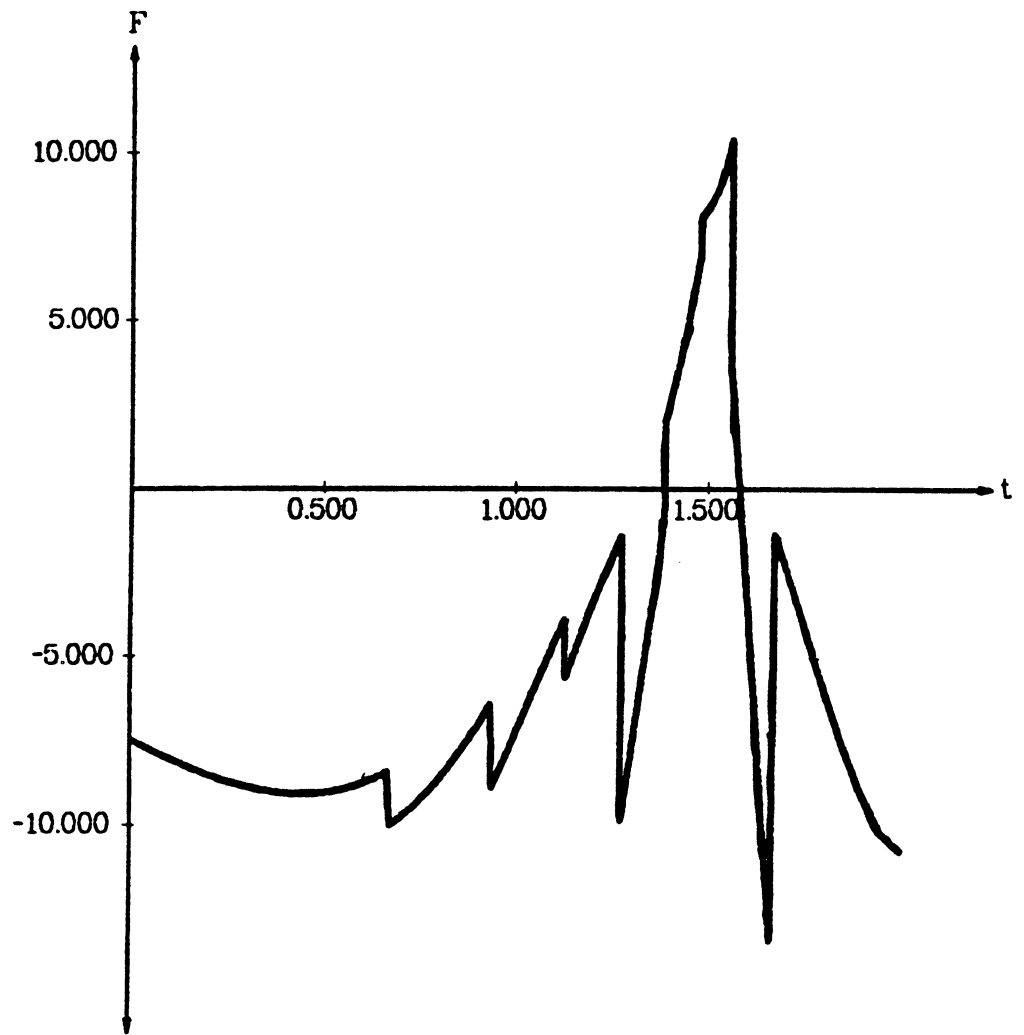


Figure 4.3.6c.  $r$  joint force vs. time,  $10 \times 10$  grid

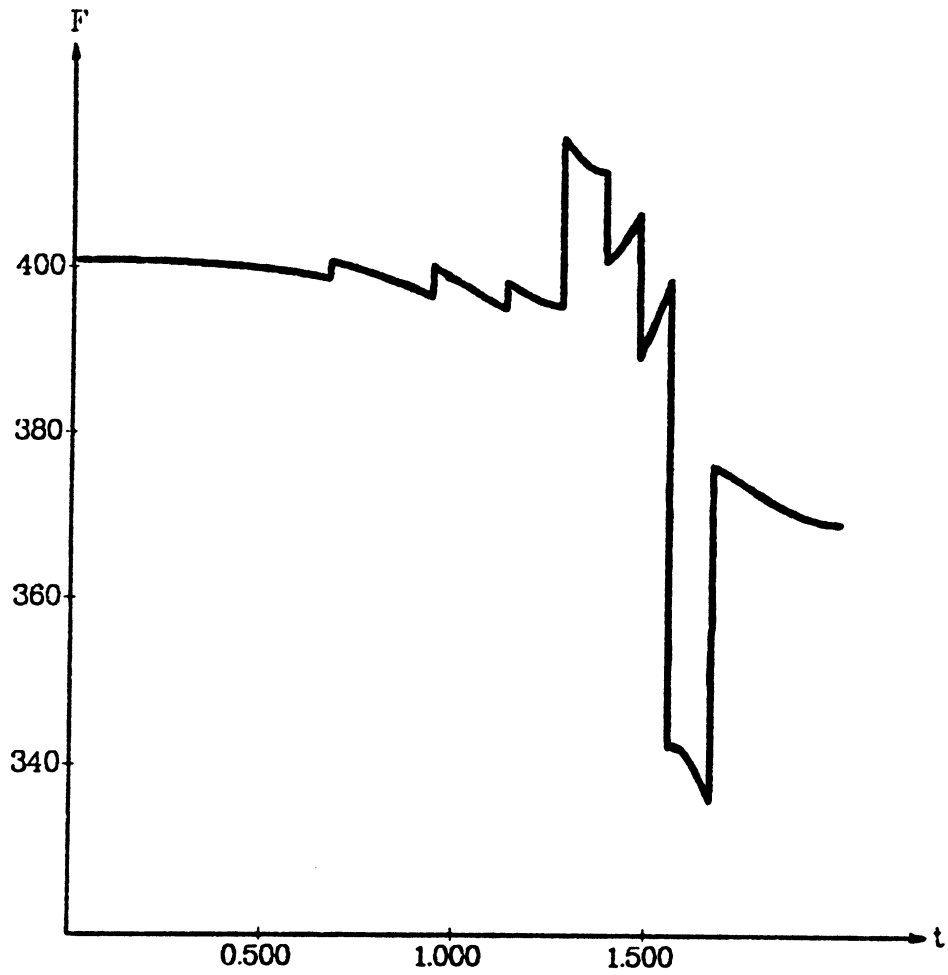


Figure 4.3.6d. z joint force vs. time, 10x10 grid

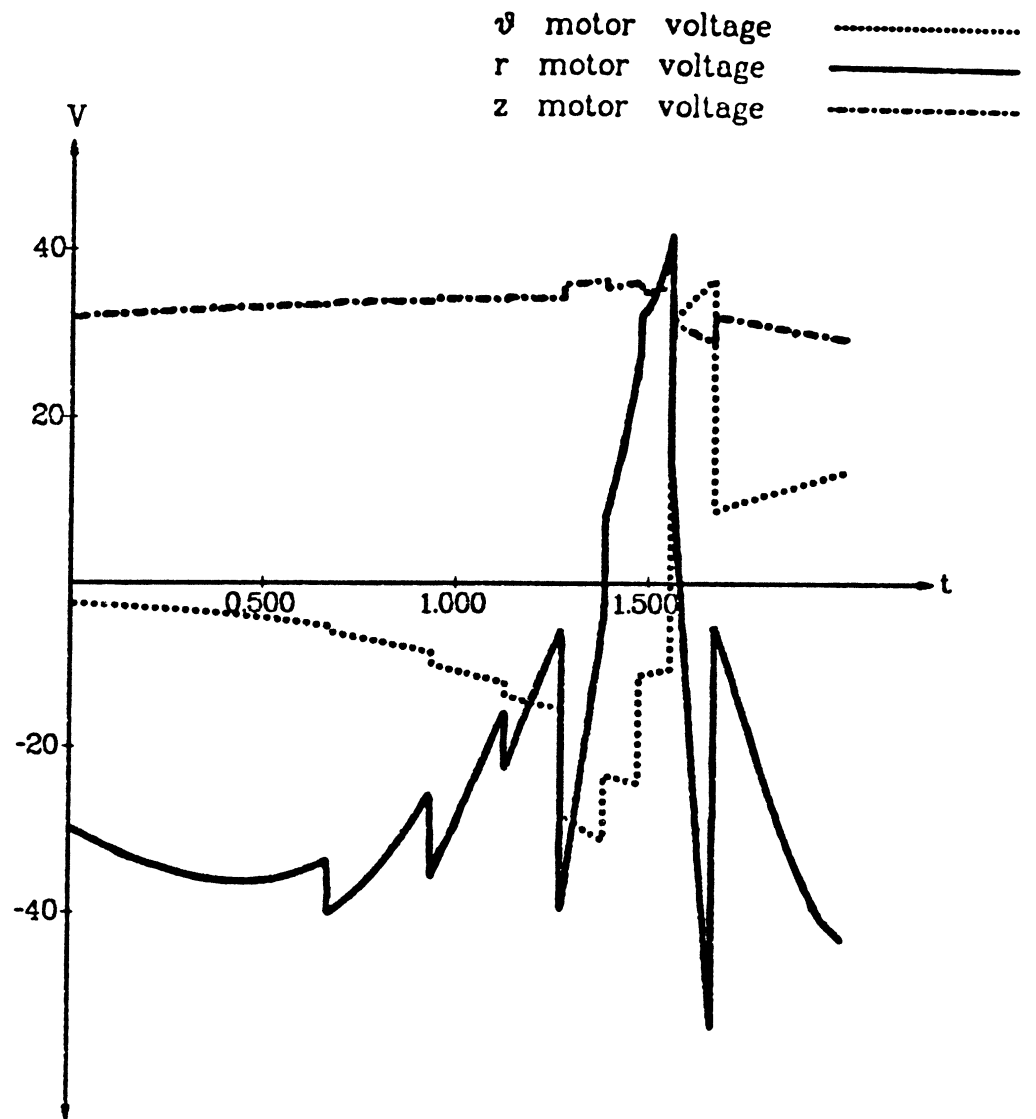


Figure 4.3.6e. Motor voltages vs. time, 10×10 grid

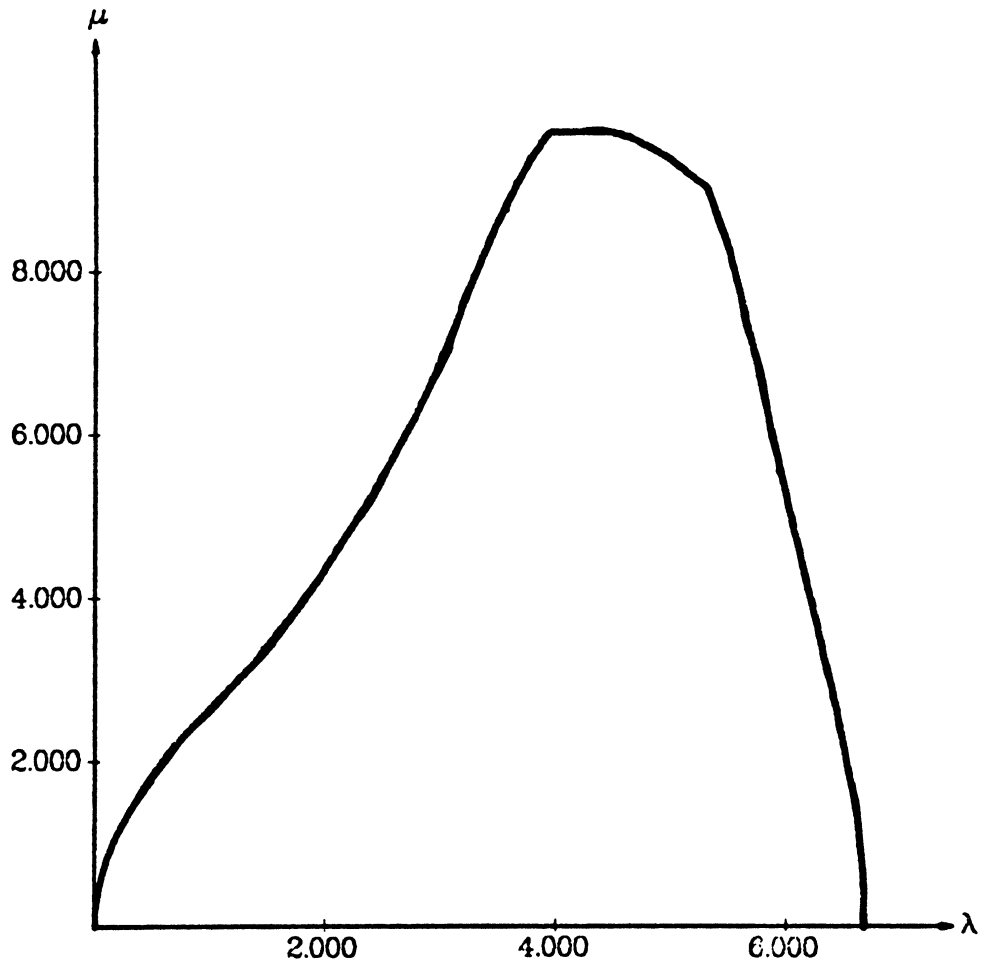


Figure 4.3.7a. Minimum time phase plot for PACS arm, 40 $\times$ 160 grid

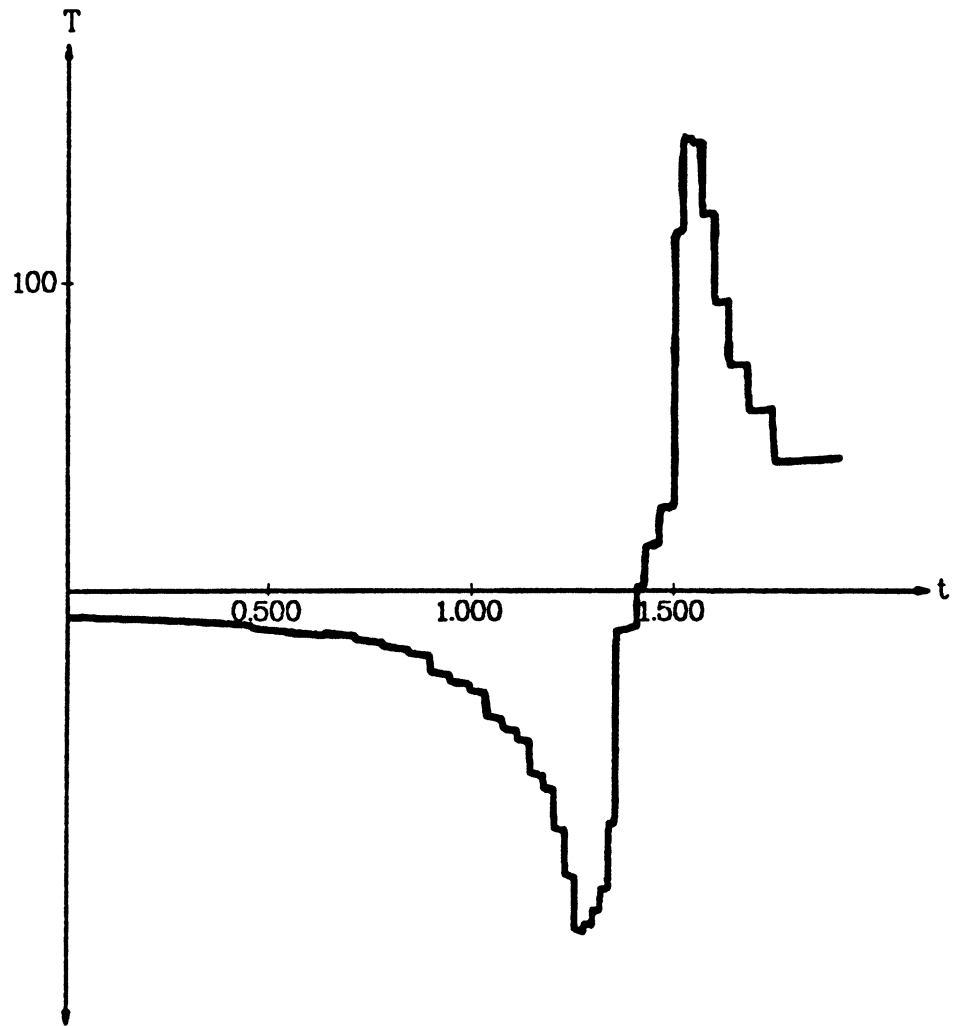


Figure 4.3.7b.  $\theta$  joint torque vs. time,  $40 \times 160$  grid



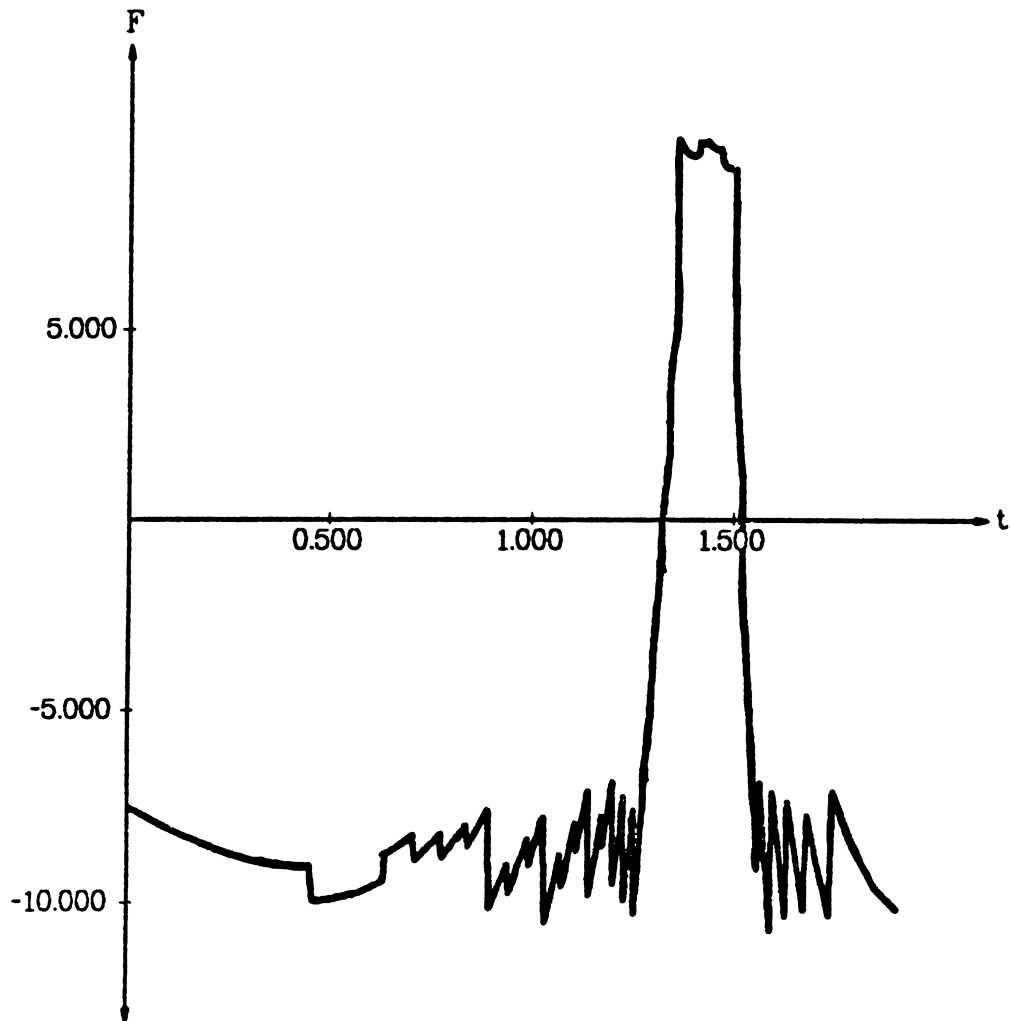


Figure 4.3.7c. *r* joint force vs. time, 40×160 grid

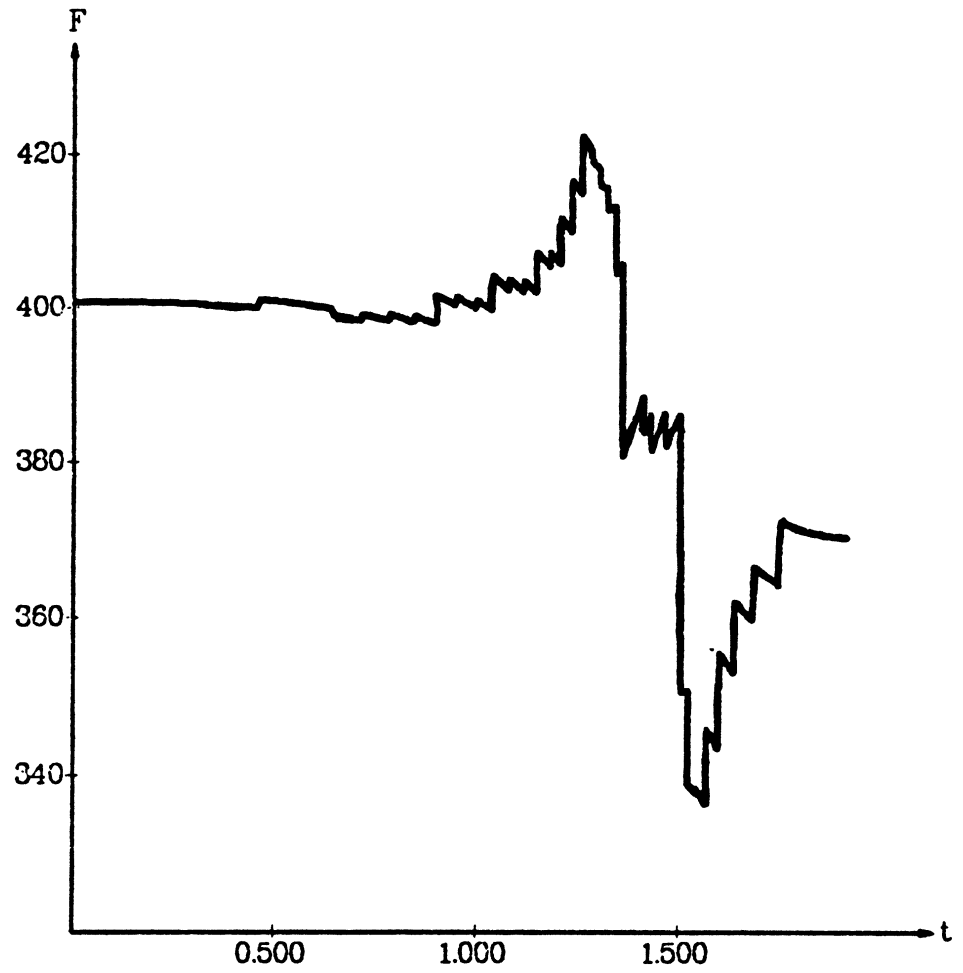


Figure 4.3.7d. z joint force vs. time, 40×160 grid

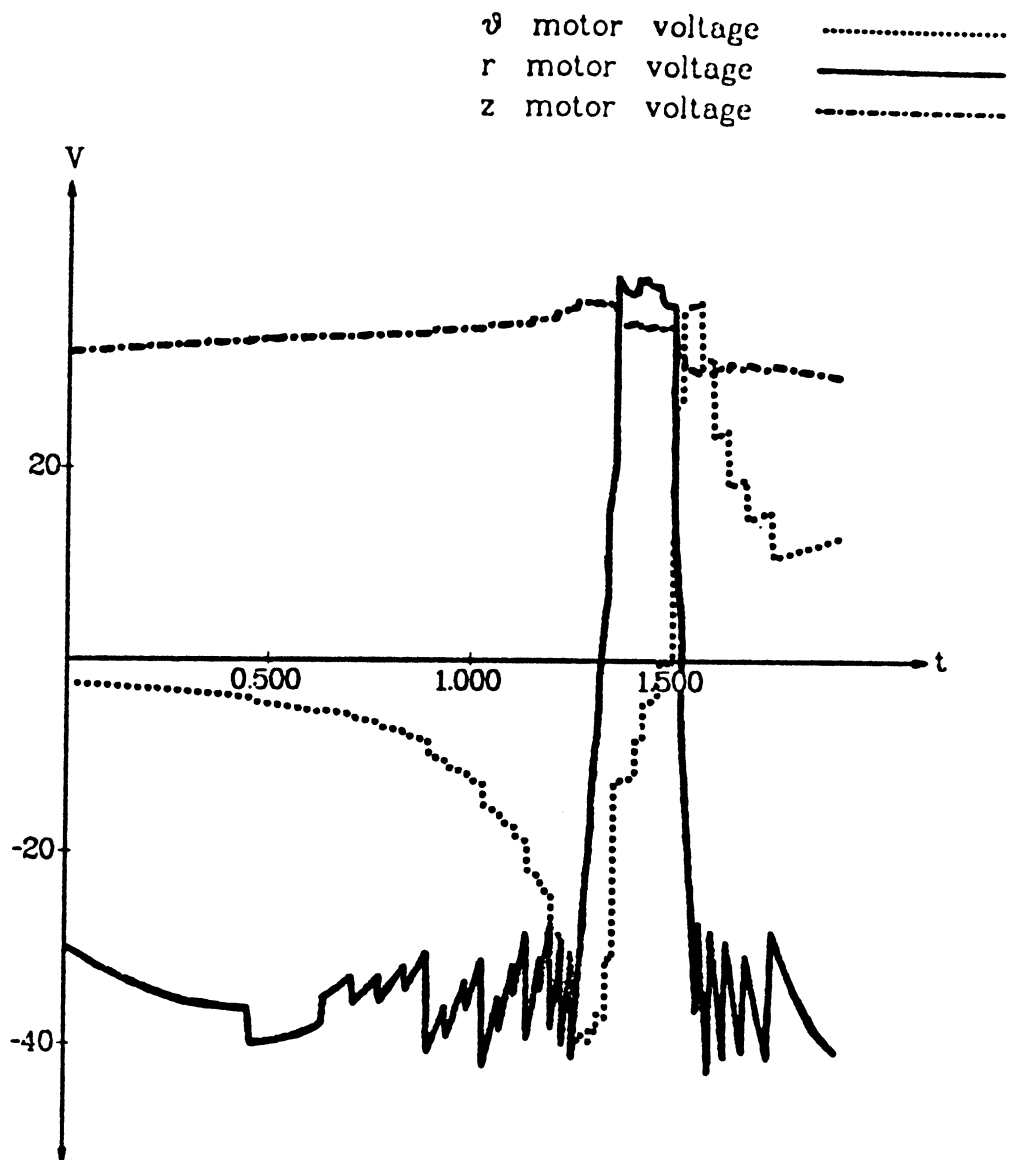


Figure 4.3.7e. Motor voltages vs. time, 40×160 grid

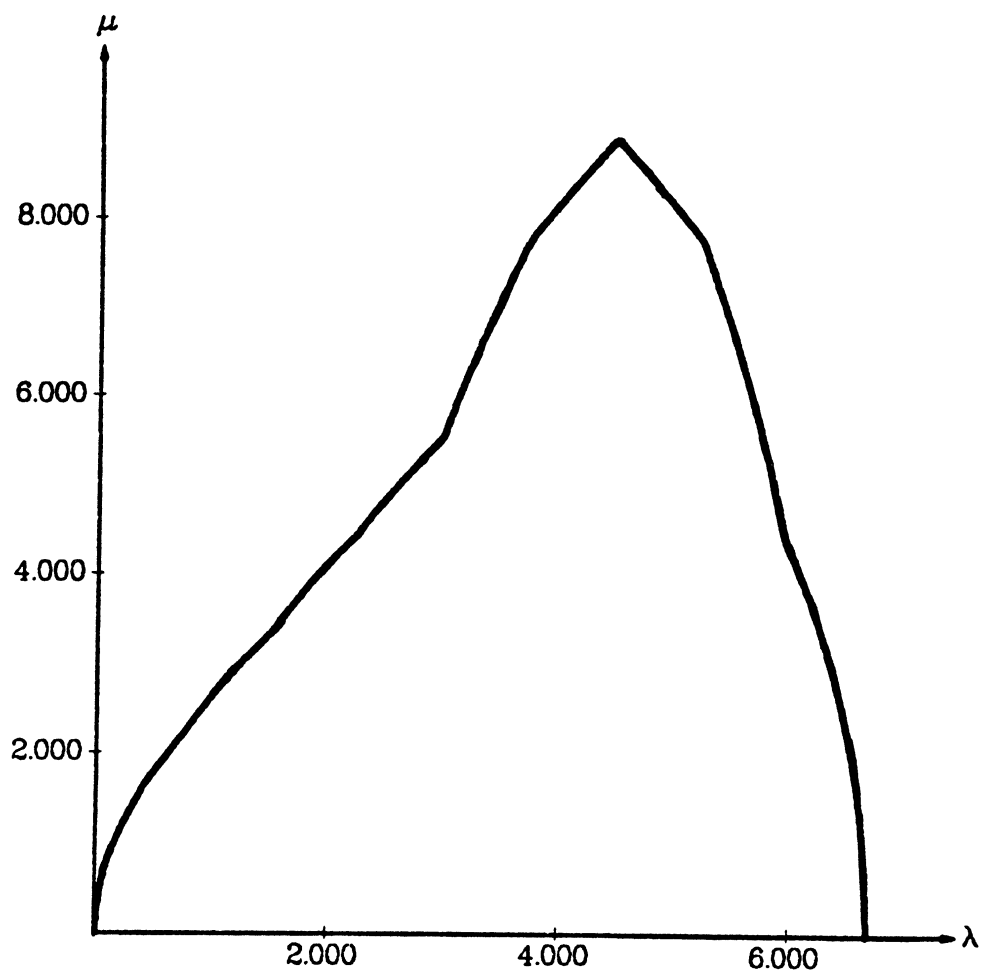


Figure 4.3.8a. Phase plot for PACS arm, minimum time-energy,  $10 \times 10$  grid

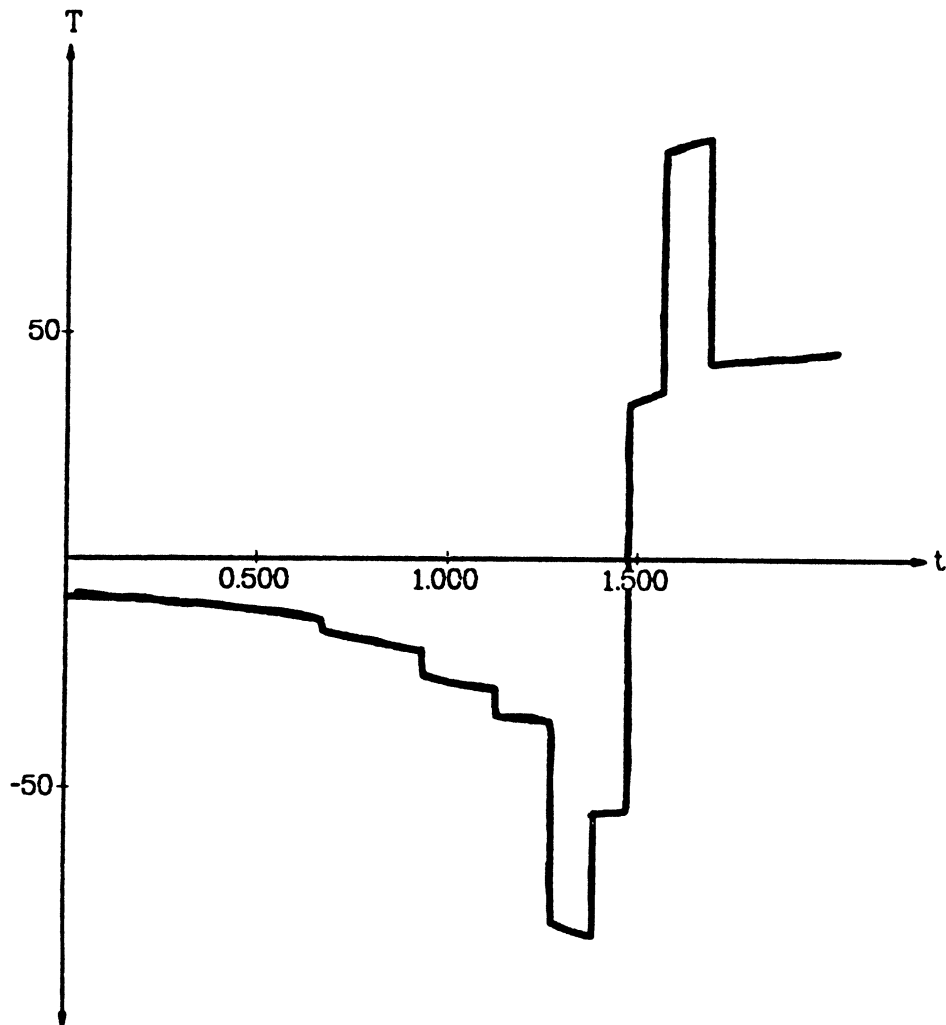


Figure 4.3.8b.  $\theta$  joint torque vs. time,  $10 \times 10$  grid

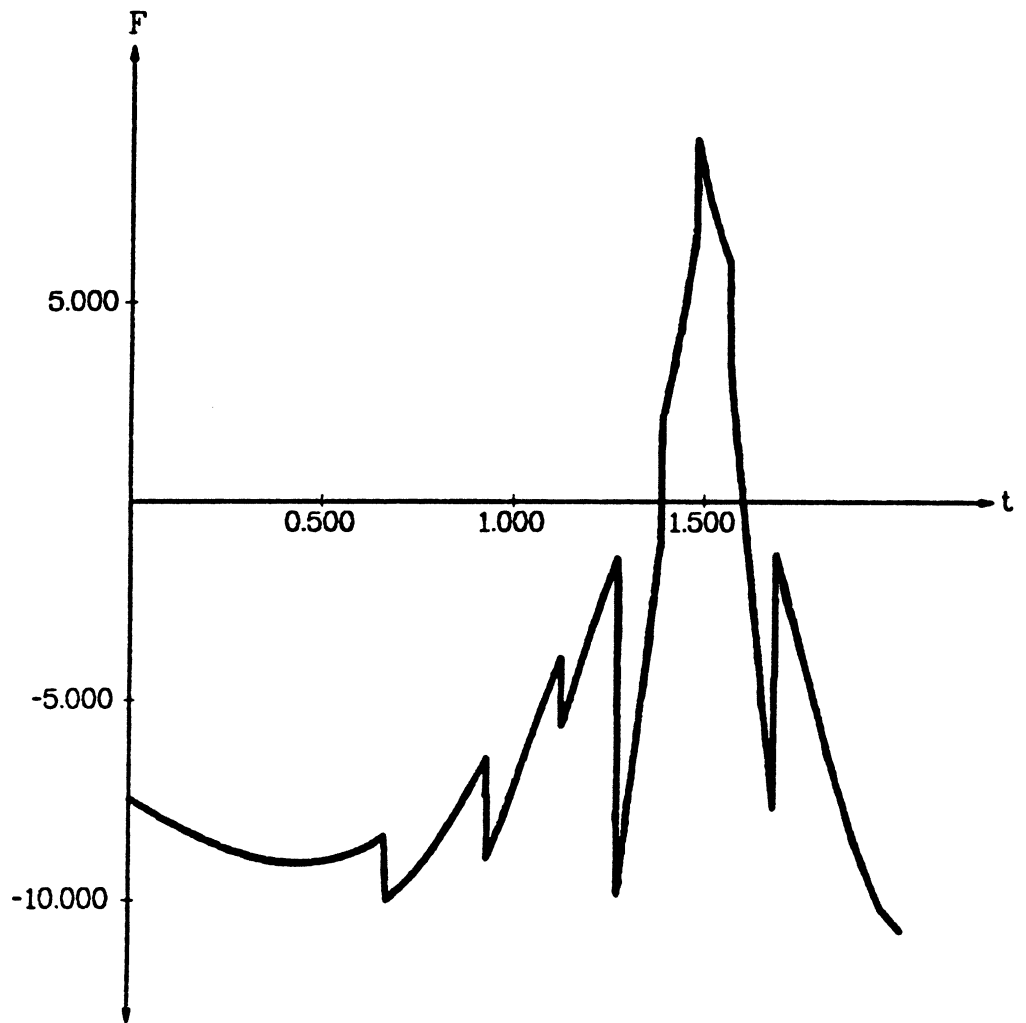


Figure 4.3.8c. *r* joint force vs. time, 10×10 grid

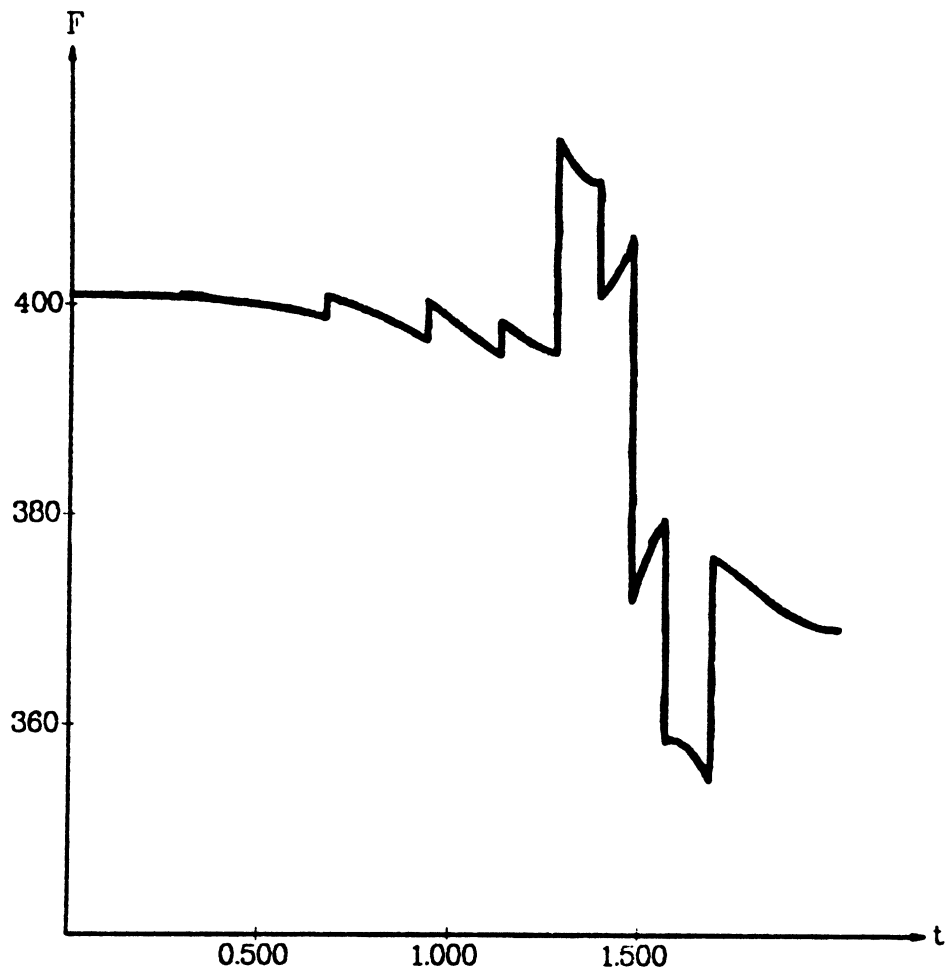


Figure 4.3.8d. z joint force vs. time, 10×10 grid

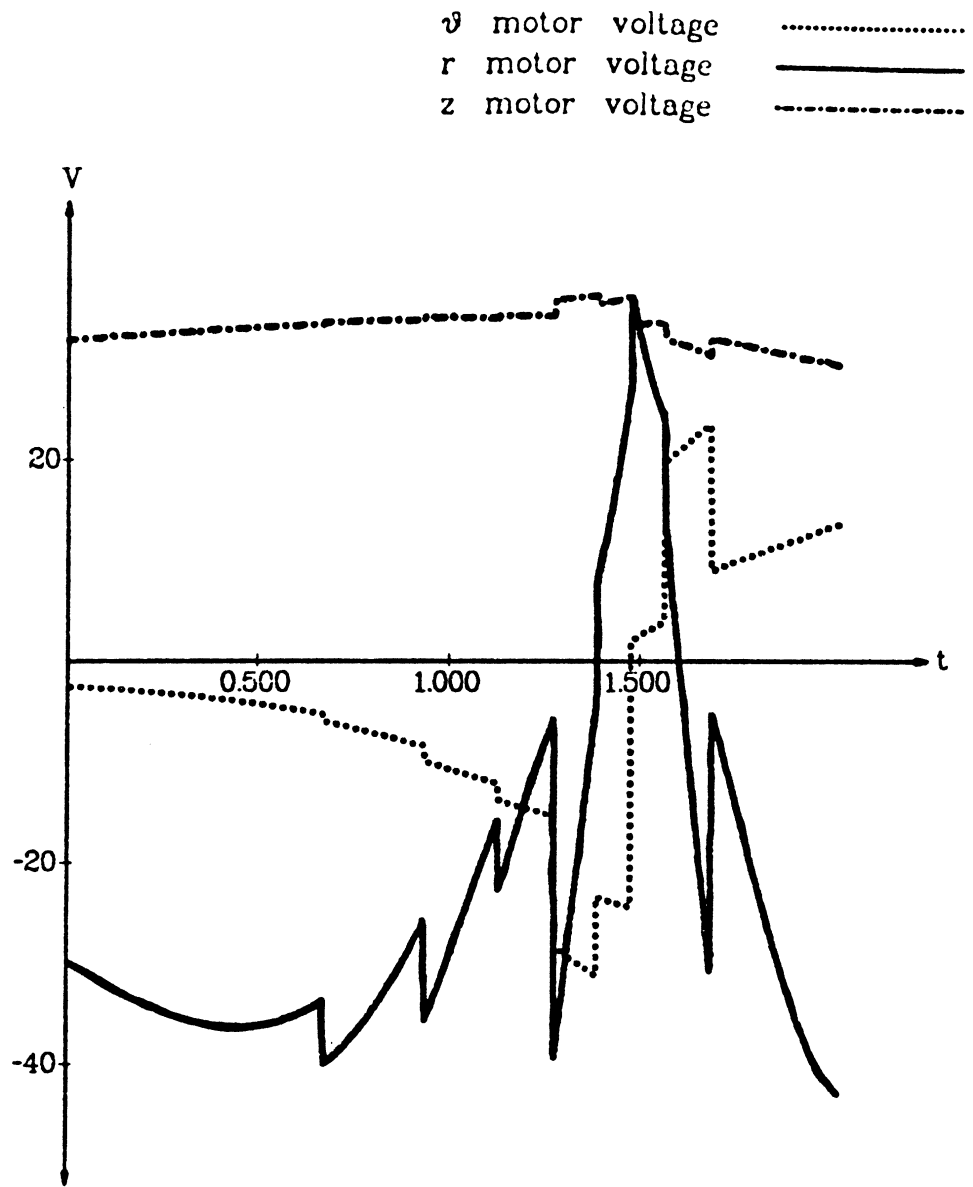


Figure 4.3.8e. Motor voltages vs. time, 10×10 grid



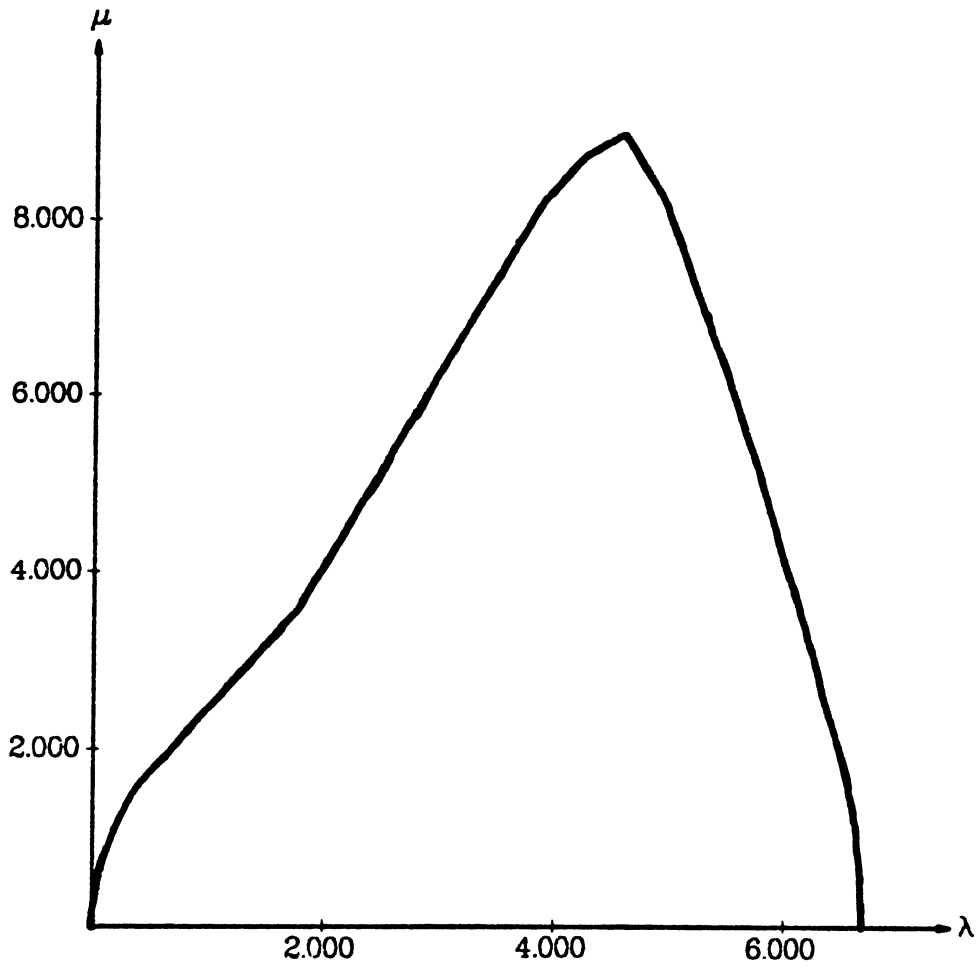


Figure 4.3.9a. Phase plot for PACS arm, minimum time-energy, 20 $\times$ 40 grid

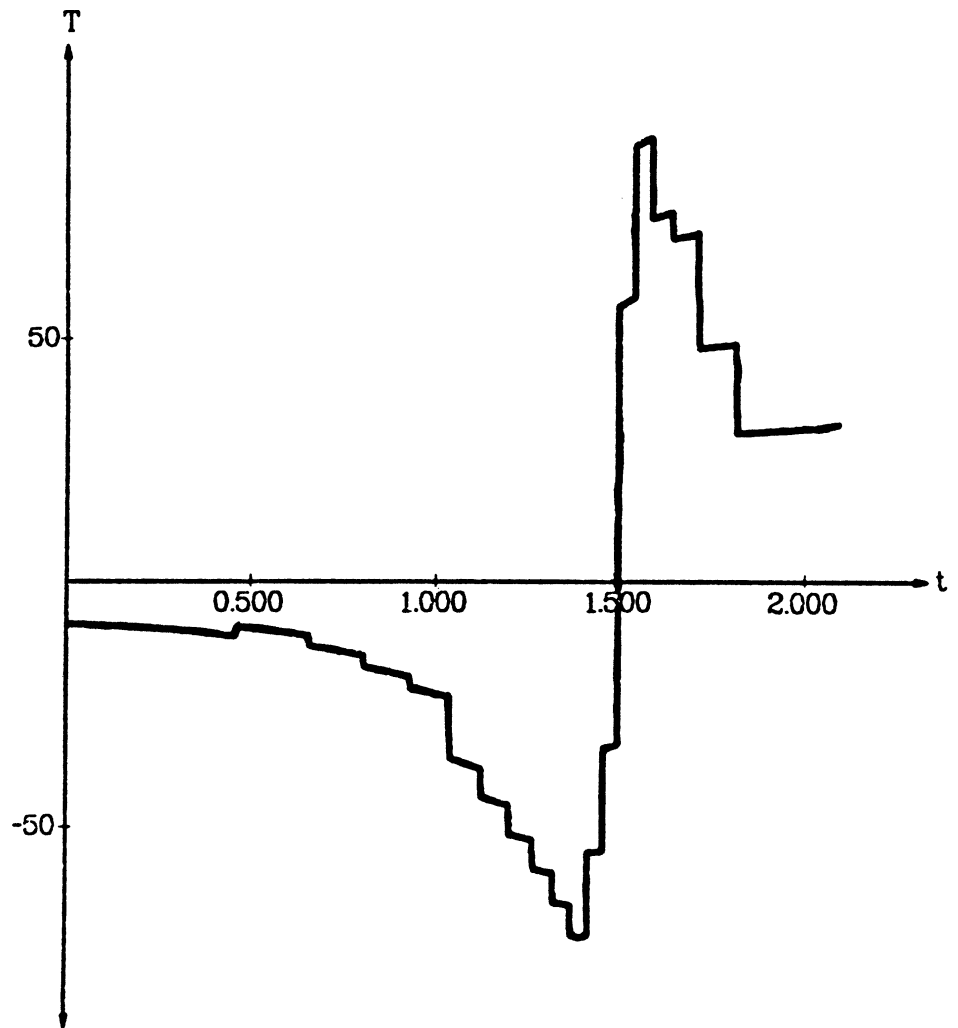


Figure 4.3.9b.  $\theta$  joint torque vs. time, 20 $\times$ 40 grid

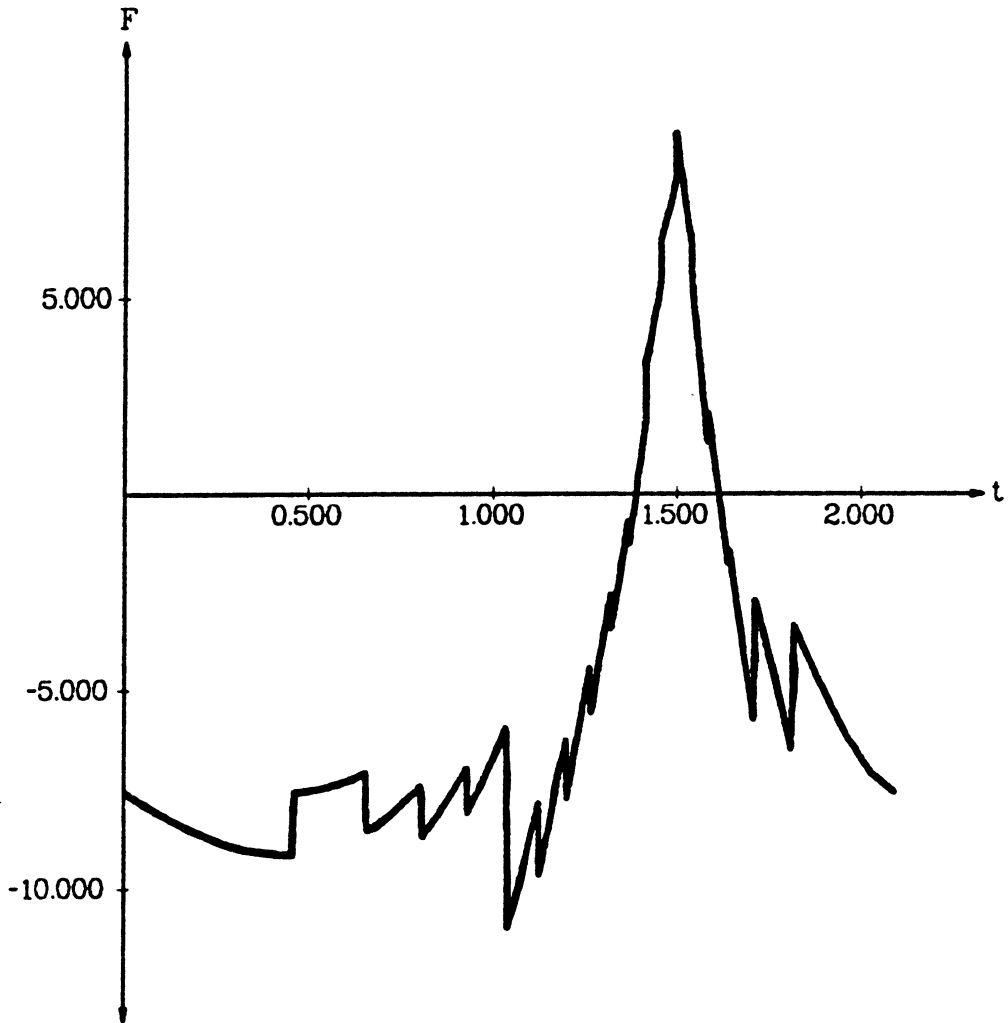


Figure 4.3.9c.  $r$  joint force vs. time,  $20 \times 40$  grid

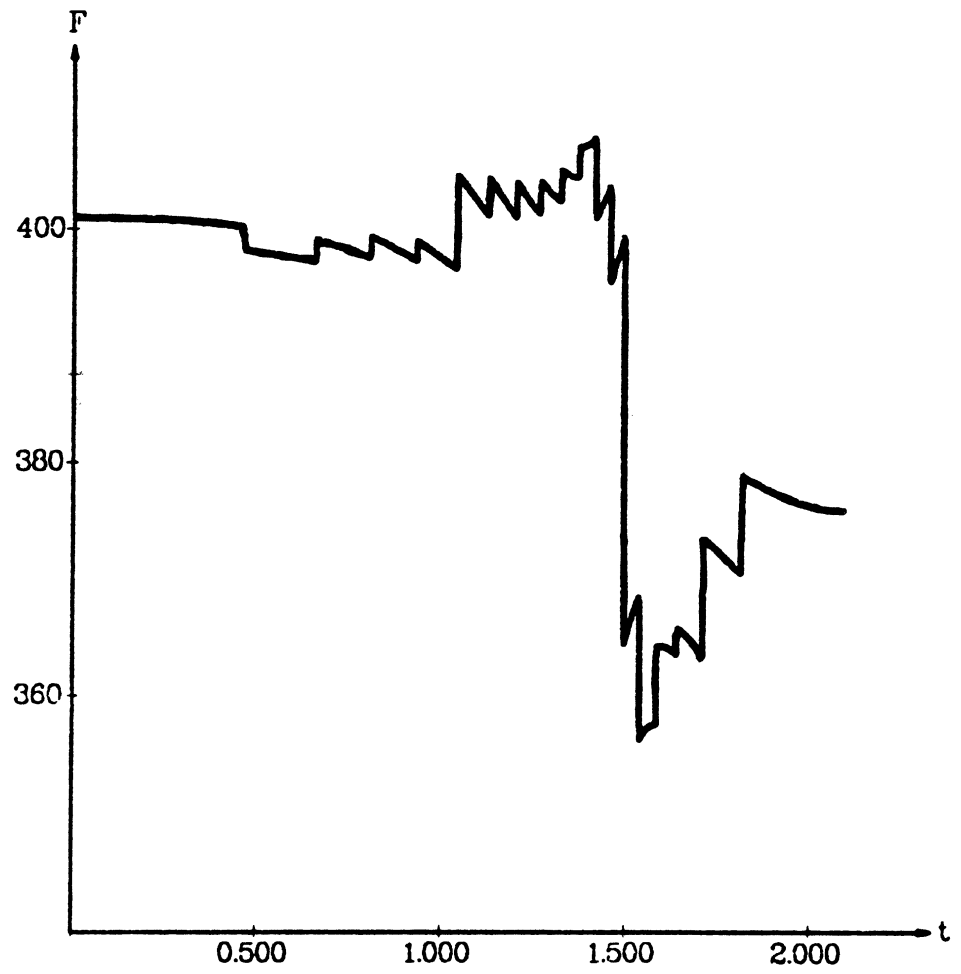


Figure 4.3.9d. z joint force vs. time, 20×40 grid

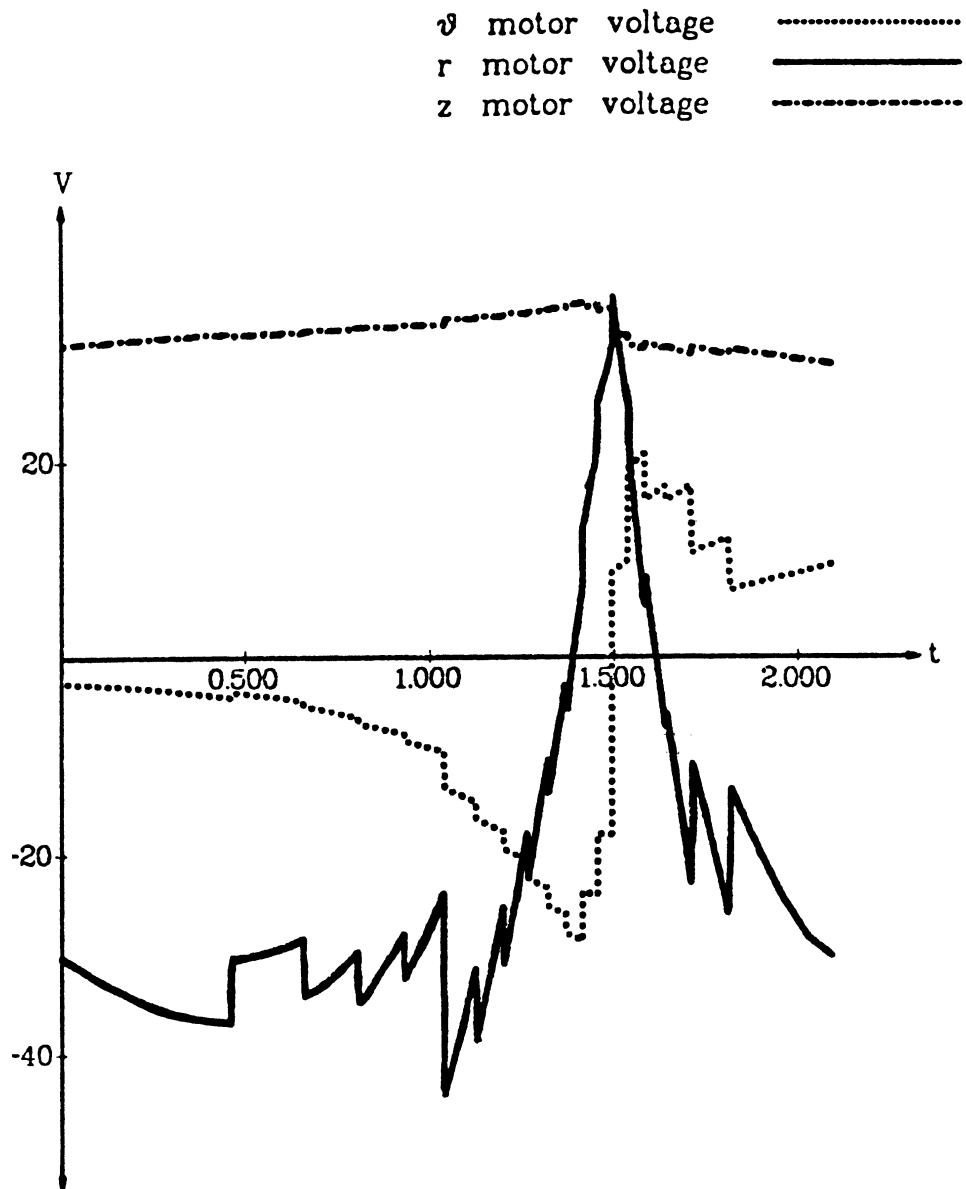


Figure 4.3.9e. Motor voltages vs. time, 20×40 grid

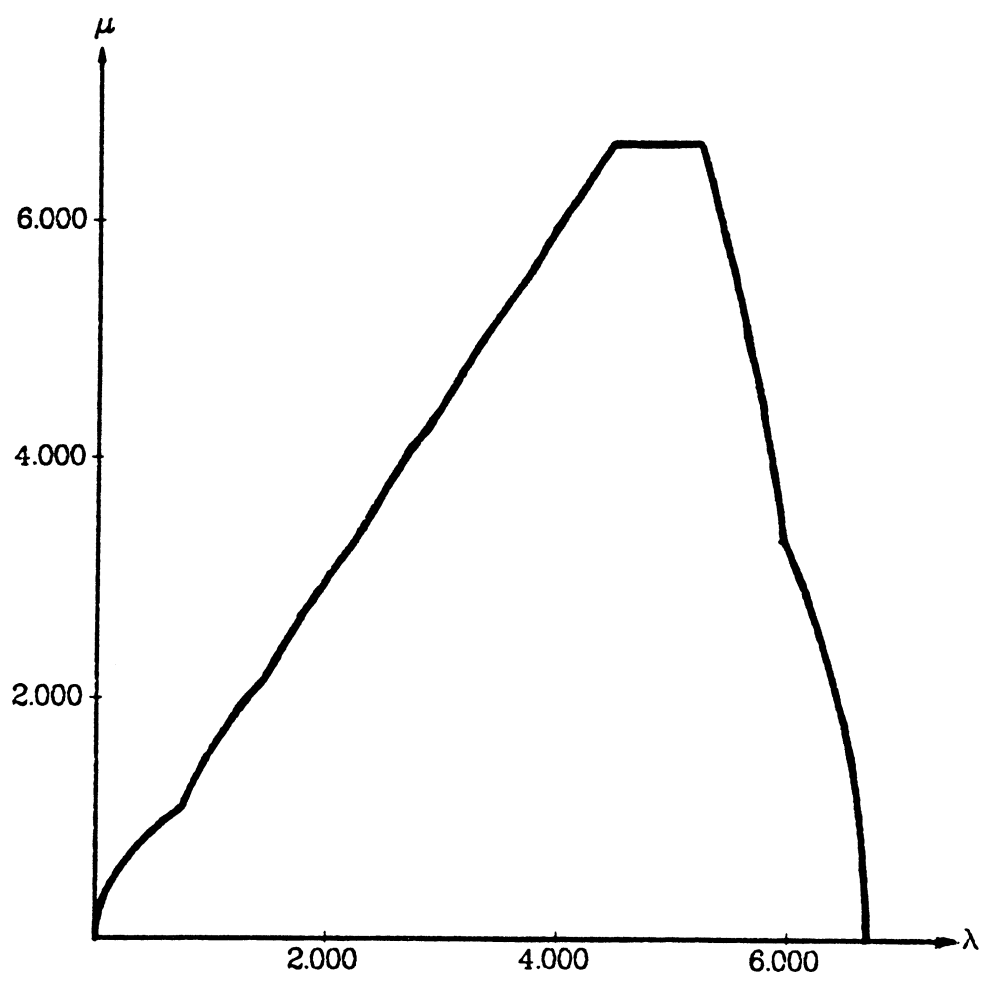


Figure 4.3.10a. Minimum time phase plot for PACS arm with power limit, 10x10 grid

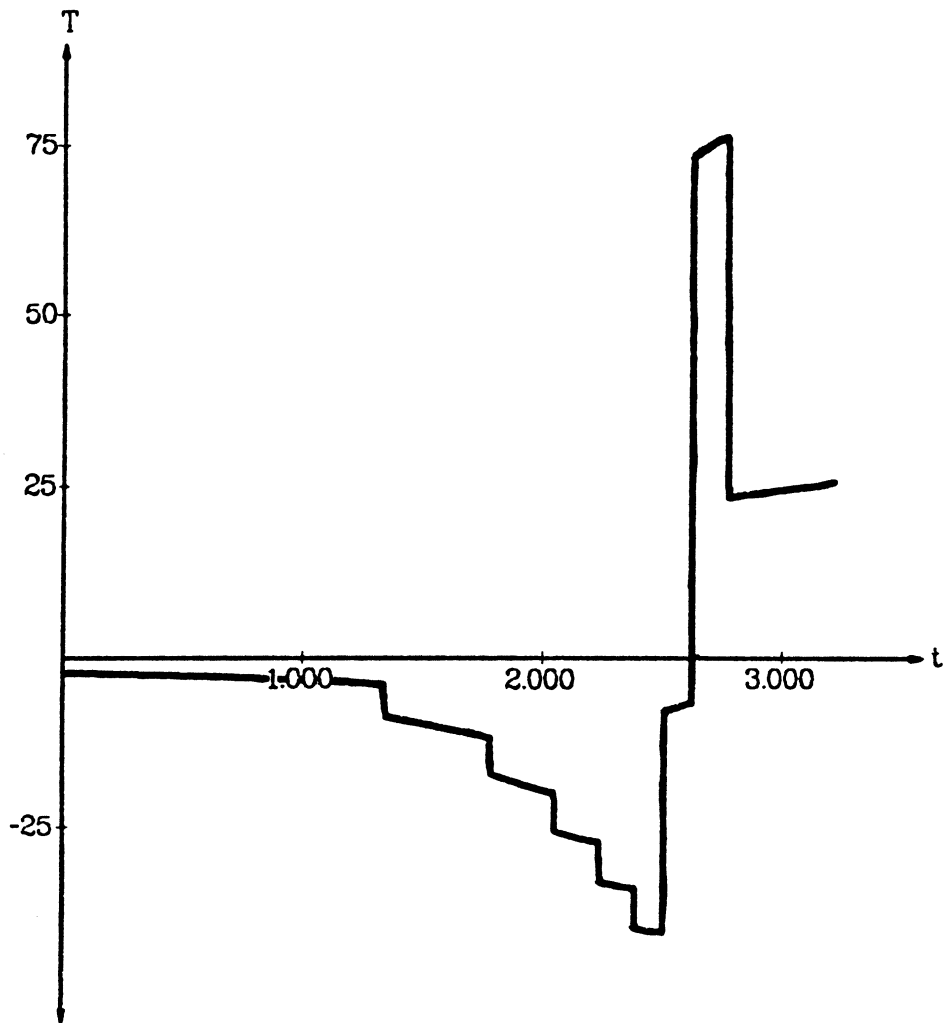


Figure 4.3.10b.  $\theta$  joint torque vs. time,  $10 \times 10$  grid

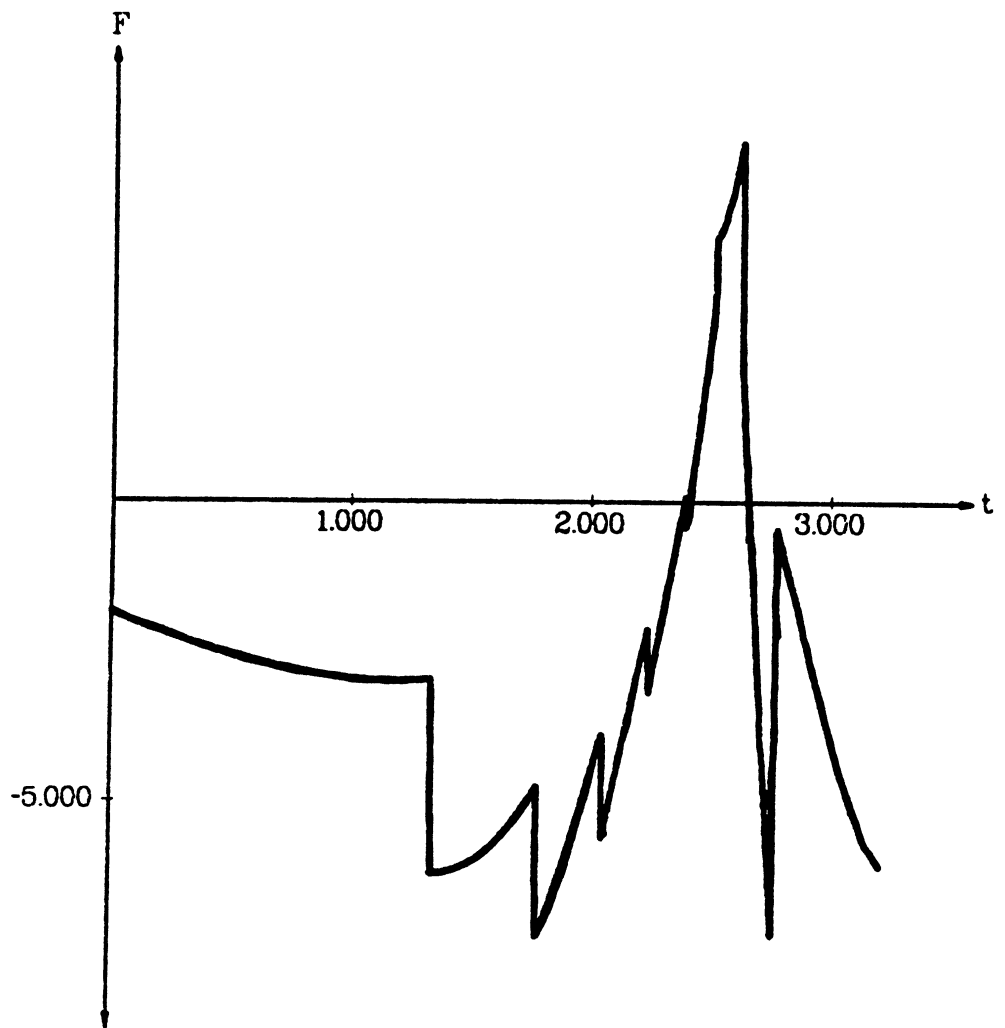


Figure 4.3.10c.  $r$  joint force vs. time,  $10 \times 10$  grid



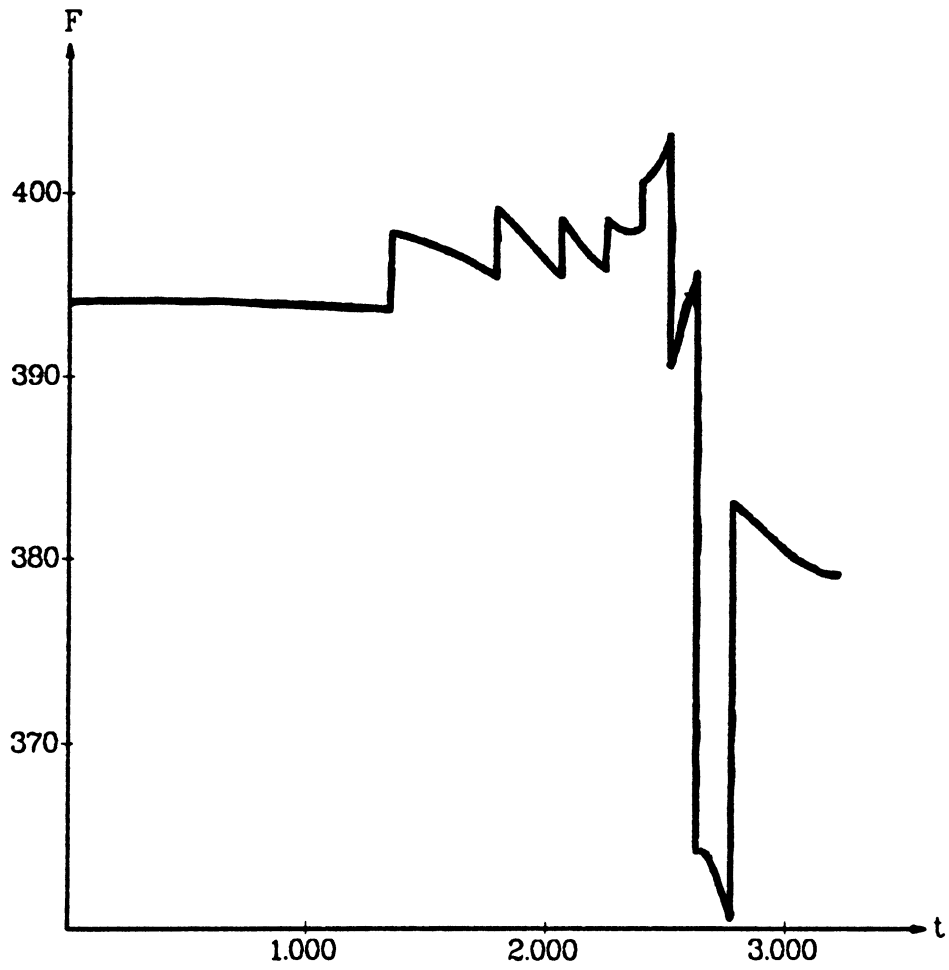


Figure 4.3.10d. z joint force vs. time, 10×10 grid

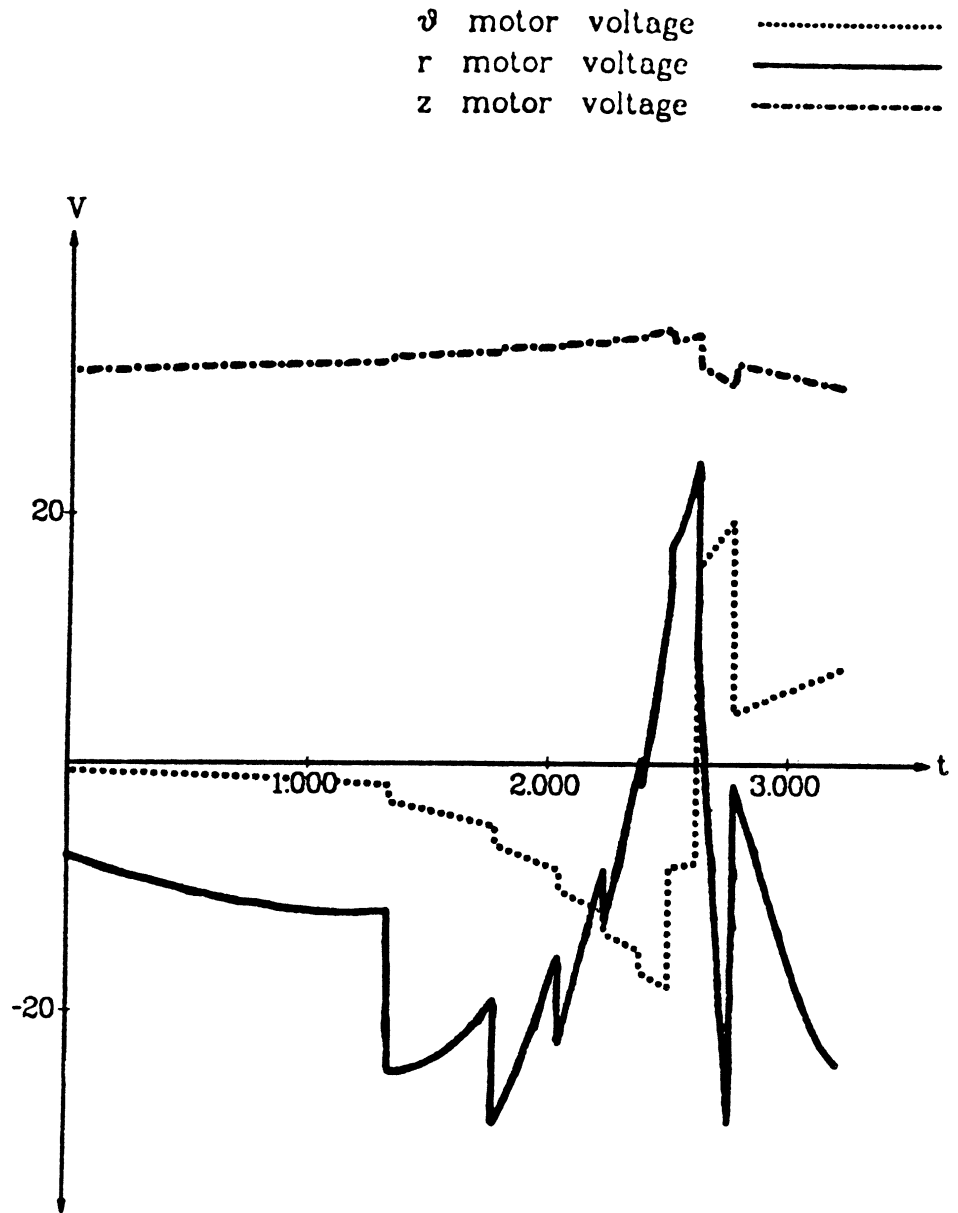


Figure 4.3.10e. Motor voltages vs. time, 10×10 grid

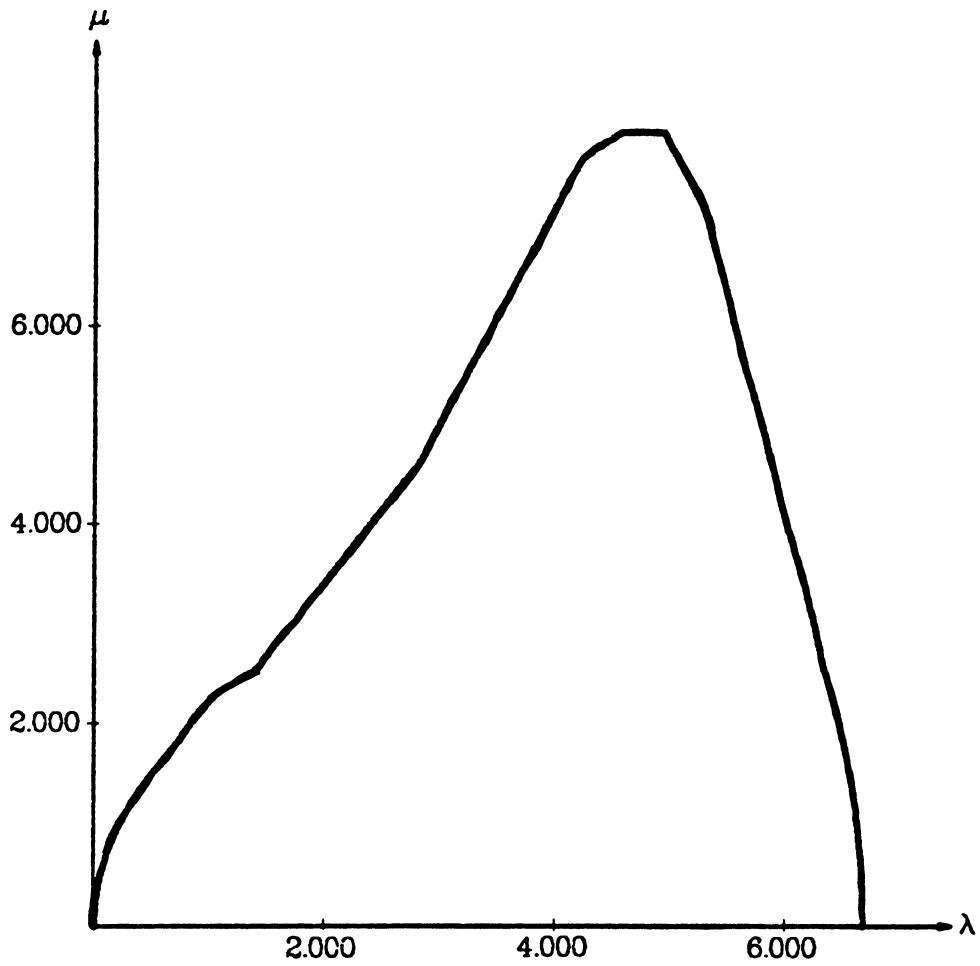


Figure 4.3.11a. Minimum time phase plot for PACS arm with power limit, 20 $\times$ 40 grid

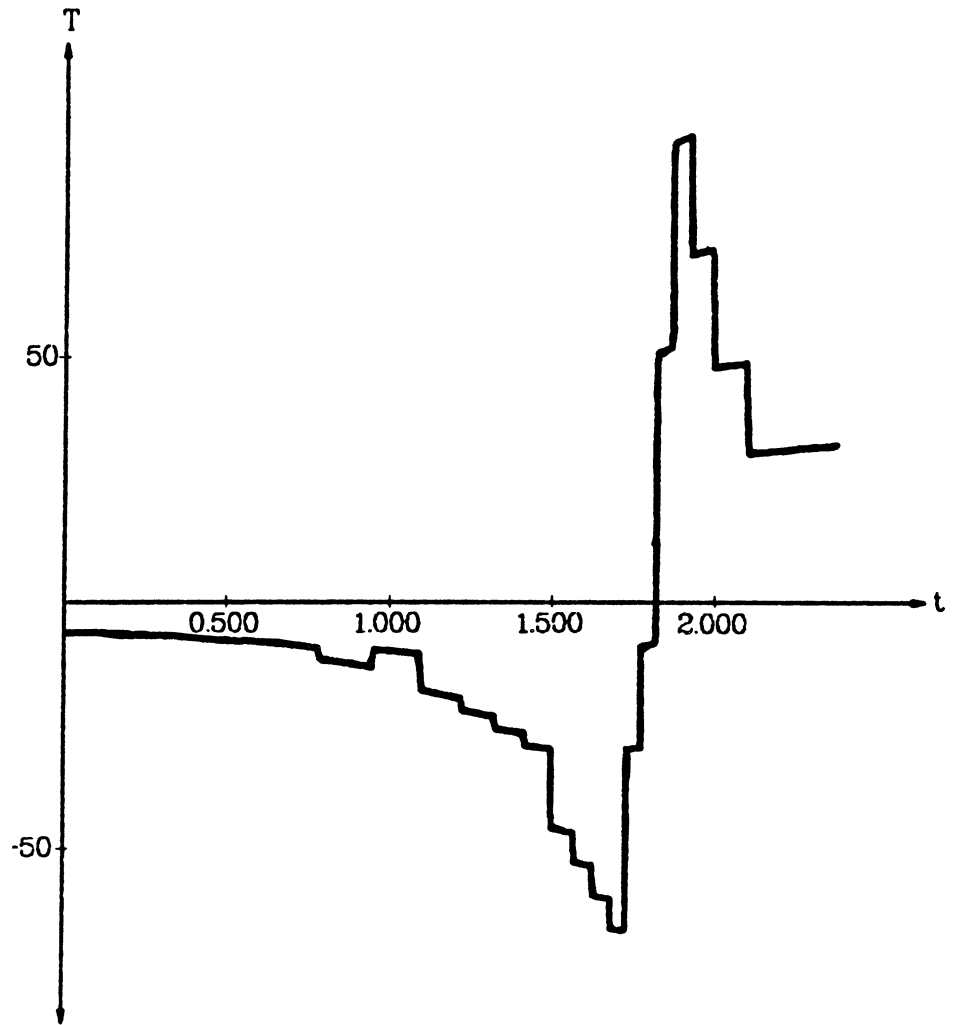


Figure 4.3.11b.  $\theta$  joint torque vs. time, 20 $\times$ 40 grid

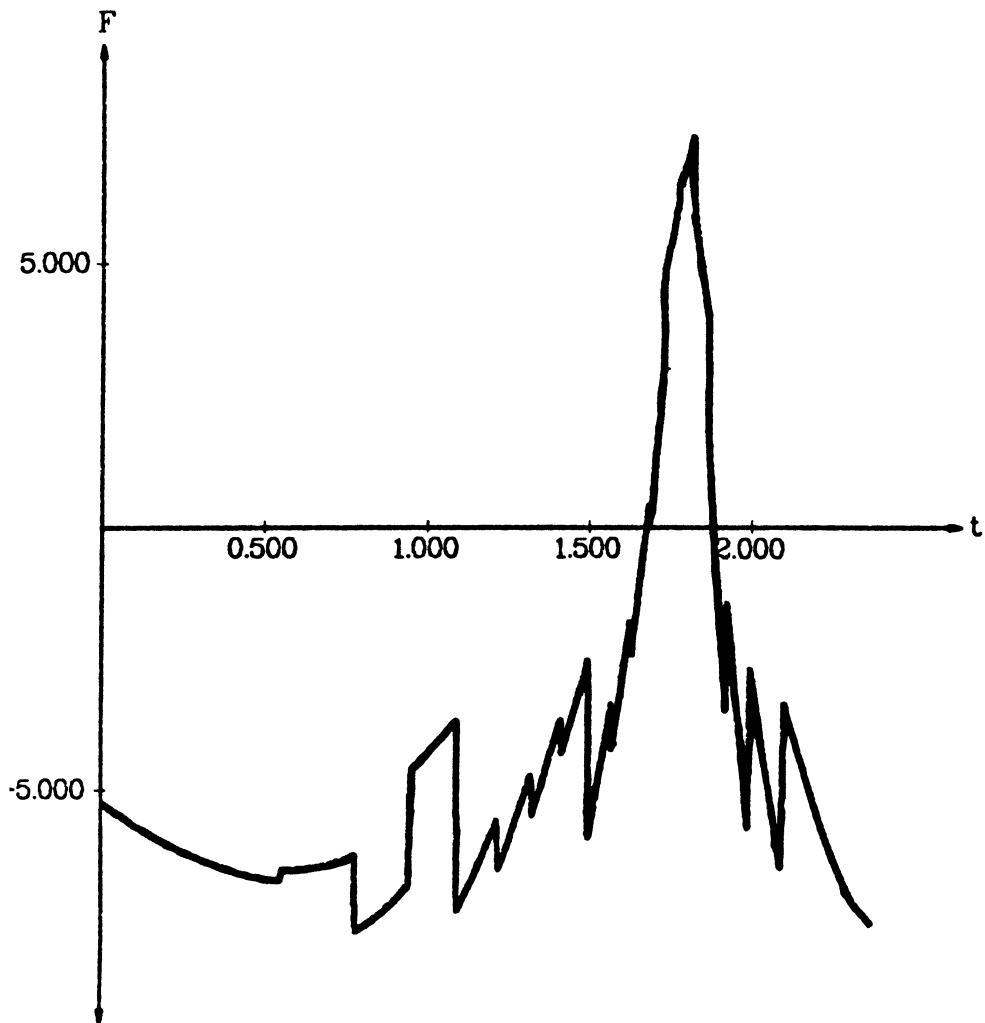


Figure 4.3.11c. *r* joint force vs. time, 20×40 grid

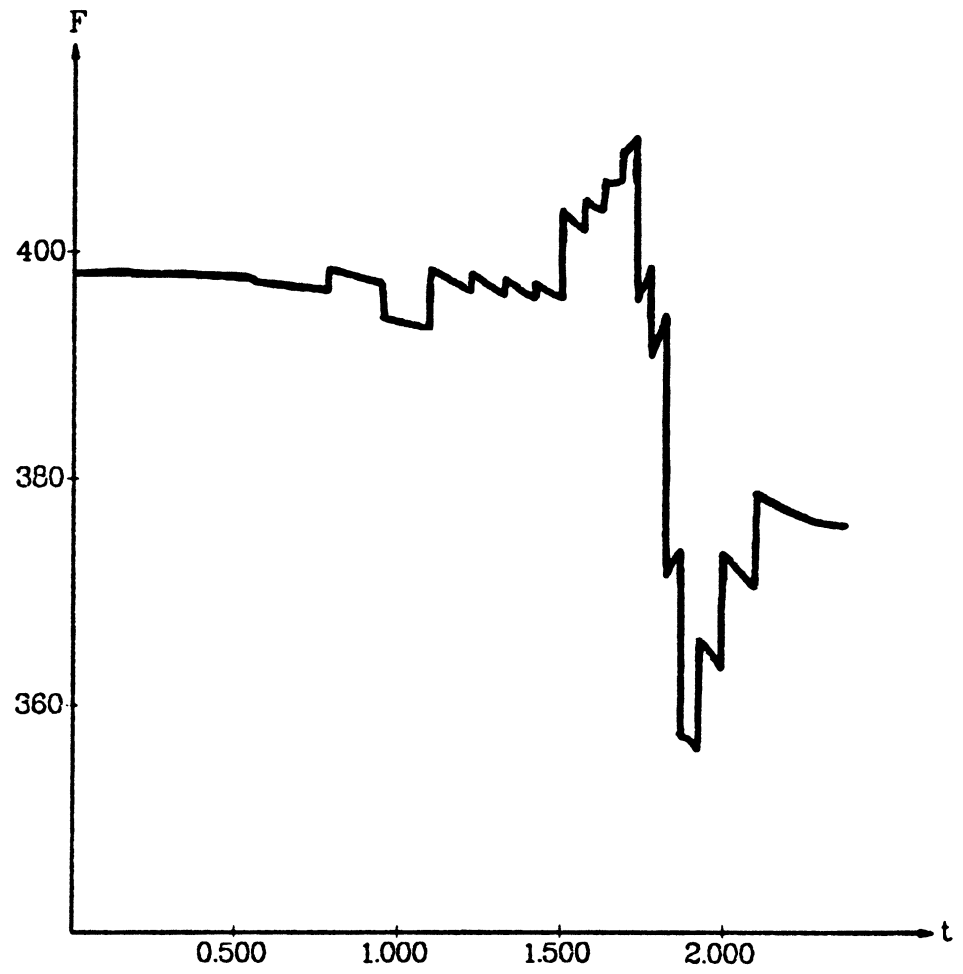


Figure 4.3.11d. z joint force vs. time, 20×40 grid

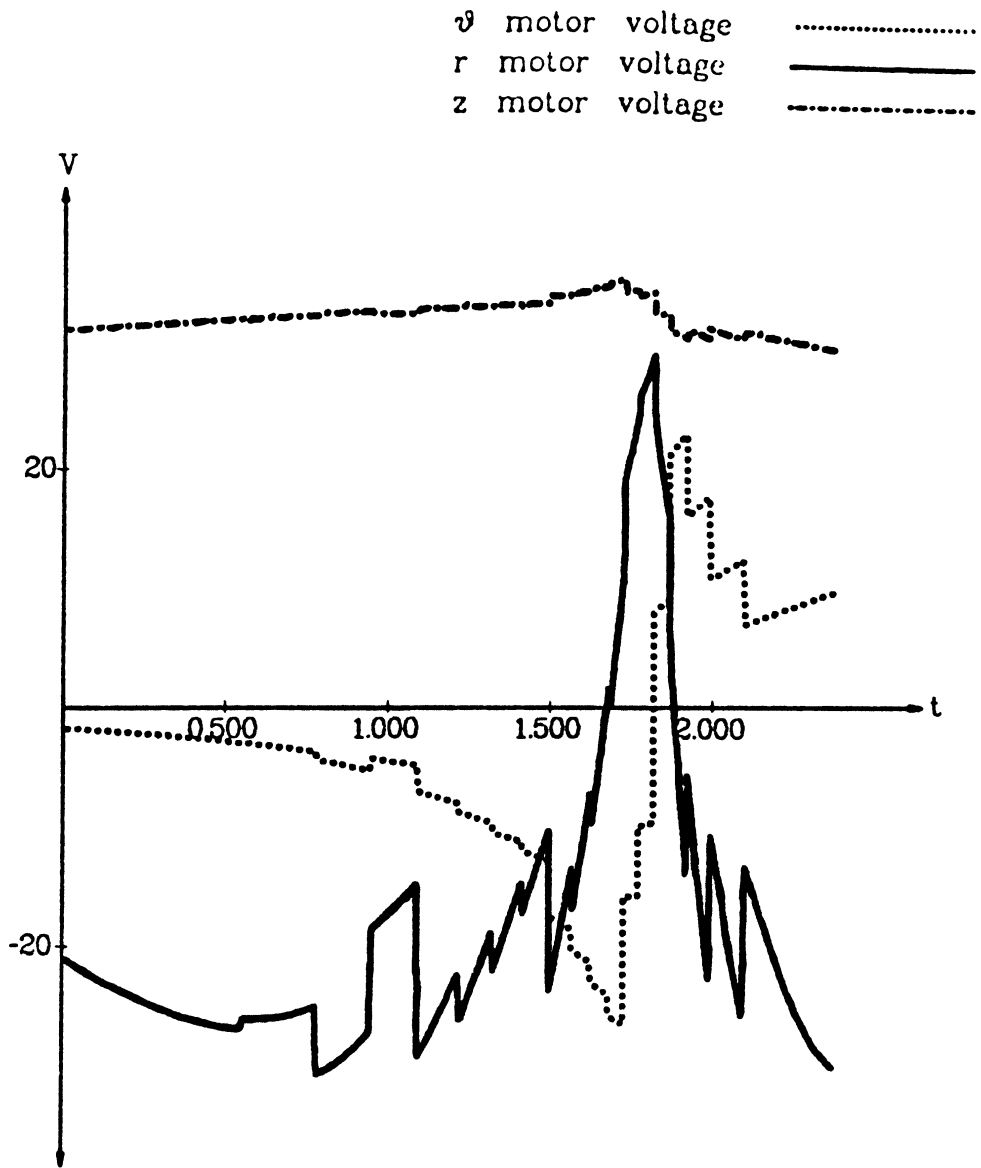


Figure 4.3.11e. Motor voltages vs. time, 20×40 grid

#### 4.4. Trajectory Planning by Iterative Improvement

It was pointed out in the previous section that the dynamic programming algorithm, particularly when used with a very fine grid, wastes a great deal of computation time by testing admissibility criteria and evaluating cost functions for sections of trajectories which are not part of, and indeed are not even close to, the optimal trajectory. It was then suggested in Section 4.3.7 that reasonable computational accuracy could be obtained from the dynamic programming algorithm by performing the dynamic programming algorithm repeatedly using a grid with a small number of divisions of the state variable, while changing the grid spacing at each iteration. The trajectory planning method proposed here, the *perturbation trajectory improvement algorithm*, or PTIA, is a variation on this theme. In addition, the PTIA may be extended to include more general classes of torque constraints. The inclusion of jerk constraints in the algorithm will be demonstrated.

At the start of PTIA, it is assumed that one phase trajectory can be found which meets all constraints. (In practice, a trajectory with zero velocity usually suffices.) This trajectory can then be perturbed to find the one with the shortest traversal time. This perturbation process is made particularly simple by the fact that minimizing time is equivalent to maximizing velocity; we wish to maximize  $\mu$ , so the phase trajectory should always be pushed upward.

##### 4.4.1. The Perturbation Trajectory Improvement Algorithm

In practice, a trajectory planner must deal with a variety of arbitrary parametric curves; two representations for curves which immediately suggest themselves are splines and simple sequences of points. As before, we choose to use the



latter representation, i.e., the curve (4.1.1) is represented as an ordered sequence of points  $(\lambda(t), \mathbf{q}(t))$ ; this proves to be the most natural representation for the application of the PTIA.

The trajectory planning process consists of assigning values of the "velocity"  $\mu$  and "acceleration"  $\dot{\mu}$  at each point. For the sake of simplicity, consider only those constraints which can be expressed in terms of position- and velocity-dependent bounds on the torque, i.e., ignore jerk constraints for the time being. Then all constraints can ultimately be given as  $\lambda$ - and  $\mu$ -dependent constraints on  $\dot{\mu}$ , or equivalently constraints on  $\frac{d\mu}{d\lambda}$ . In terms of the  $(\lambda, \mu)$  plot, each point is assigned a set of allowable slopes. In the discrete approximation, this sets limits on the differences between the values of  $\mu$  at adjacent points. The process of trajectory planning requires that the initial and final points of the curve have zero velocity (or some other fixed velocity) and that the velocities at all the intermediate points be as large as possible, consistent with the constraint that the velocities at neighboring points not differ too much.

One approach to the solution of this problem is to try to push the speed higher at each individual point. The value of  $\mu$  can be pushed higher at each point in succession until none of the velocities can be made any larger. (This is just component-wise optimization, where the infinite-dimensional space of functions  $\mu(\lambda)$  has been approximated by a finite-dimensional vector of real numbers.) If we call this Algorithm A, then we have

Algorithm A:

- 1) Set all velocities to values which are realizable (usually all zeroes).
- 2) Push each intermediate point of the curve as high as possible consistent with

the slope constraints.

3) If any of the velocities were changed in step 2, go back to step 2, otherwise exit.

As a practical matter, the search required to find the highest possible velocity in step 2 of Algorithm A may be fairly expensive, especially since it may be repeated many times for a single point. A simpler approach is to just try adding a particular increment to each velocity, and then make the increment smaller on successive passes of the algorithm. This gives

Algorithm A':

- 1) Set all velocities to values which are realizable (usually all zeroes).
- 2) Set the current increment to some large value.
- 3) Push each intermediate point of the curve up by an amount equal to the current increment, if this is consistent with the slope constraints.
- 4) If any of the velocities were changed in step 3, go back to step 3.
- 5) If the current increment is smaller than the desired tolerance, stop. Otherwise halve the increment and go to 3.

Algorithm A' is really just a combination of gradient and binary search techniques. The direction in which the curve must move (i.e., the gradient direction) is known *a priori*, since increasing the velocity always decreases the traversal time, and the amount of the change is successively halved, as in a binary search, until some desired accuracy is achieved. Algorithm A' is very simple, except possibly for the slope constraint check required in step 3. This requires a knowledge of the dynamics and actuator characteristics of the robot. However, this check is a simple "go/no go" check, and can be isolated as a single function call. (Hereafter this function will be

called the *constraint function*.) Hence the trajectory planner can be used with other robots by changing a single, though possibly complicated, function.

An important characteristic of the constraint function is *locality*. In the case discussed above, the constraints are expressed in terms of  $\lambda$ ,  $\mu$ , and  $\frac{d\mu}{d\lambda}$ . We need two points to determine the slope  $\frac{d\mu}{d\lambda}$ , so the constraint depends only upon two points. Therefore when a point of the curve has its  $\mu$  value changed, it is constrained only by the two adjacent points; the rest of the curve has no influence. This allows much calculation to proceed in parallel. Step 3 of Algorithm A' can be divided into two steps, one which increments the odd numbered points and one which increments the even numbered ones. Since the even numbered points stay the same while the odd numbered ones are being incremented, and vice versa, the points either side of the incremented points remain stationary, so that the constraint checks are valid. (If all points were tested simultaneously, then it is possible, for example, to increment two adjacent points; since in each case the constraint check would be made on the assumption that the other point was remaining stationary, it is possible that the new configuration would not meet the required constraints.)

It is easily seen that the process in Algorithm A' can be extended to more complicated constraints. For example, constraints on the jerk (the derivative of the acceleration) only require a more complicated constraint function. Of course in this case the constraint function needs three points to calculate second derivatives of the speed. Thus the constraints on a single point will be functions of *two* points either side of the point being checked, rather than one point. This affects the degree of parallelism which can be achieved; step 3 would require three passes instead of two.

It also may make the algorithm diverge as the number of points on the curve increases, as will be seen later from the numerical examples.

As a simple illustration of how the algorithm works, consider a simple one-dimensional problem. Suppose we wish to move an object of mass  $m$  from  $x=0$  to  $x=4$ . Further suppose that there is no friction, and that there are constant bounds on the magnitude of the applied force. There will be only one parametric function  $f$ , which may be taken to be the identity function, so that  $\lambda=x$ . We then have

$$F = ma = m \frac{d^2 x}{dt^2} = m \frac{d^2 \lambda}{dt^2} = m \frac{d \mu}{dt} = m \frac{d \mu}{d \lambda} \frac{d \lambda}{dt} = m \mu \frac{d \mu}{d \lambda}.$$

If we consider  $\lambda$ -intervals of length 1, then the discrete approximation to the parameterized "curve" will have 5 points. The acceleration  $\ddot{\mu} = \mu \frac{d \mu}{d \lambda}$  can be approximated as

$$\mu \frac{d \mu}{d \lambda} \approx \mu \frac{\mu_{i+1} - \mu_i}{\lambda_{i+1} - \lambda_i} \approx \frac{\mu_{i+1} + \mu_i}{2} \cdot \frac{\mu_{i+1} - \mu_i}{\lambda_{i+1} - \lambda_i} = \frac{\mu_{i+1}^2 - \mu_i^2}{2(\lambda_{i+1} - \lambda_i)}$$

The torque constraints then become

$$F_{\max} \geq |F| = m |a| = m \left| \frac{\mu_{i+1}^2 - \mu_i^2}{2(\lambda_{i+1} - \lambda_i)} \right|$$

If we use  $m = 1$ ,  $F_{\max} = 2$ , and  $\lambda_{i+1} - \lambda_i = 1$ , this reduces to

$$|\mu_{i+1}^2 - \mu_i^2| \leq 4.$$

Now consider what happens if Algorithm A is applied. We may look at the intermediate points of the curve in sequence. First, point 1 can be raised by 2, since the adjacent points have  $\mu$  values of zero, and  $|2^2 - 0^2| = 4$ . Raising the middle point, point 2, we are constrained by the fact that  $\mu_3 = 0$ , which limits  $\mu_2$  to 2 also. Like-

wise, we may change  $\mu_3$  to 2. This completes step 2 of Algorithm A. Since some of the  $\mu$  values changed, we try to increase them again. This time only point 2 can be raised, giving a value of  $\mu_2 = 2\sqrt{2}$ . On the next pass, no  $\mu$  values change, so Algorithm A terminates. It is easily verified that the solution obtained from Algorithm A is indeed the optimal solution to the discretized problem. (Figure 4.4.1 shows the discretized trajectory after passes zero, one, and two of Algorithm A.)

Now look at what happens when we use Algorithm A'. Say we start with an increment of 2. Then the result of the first pass of Algorithm A' is the same as the result of the first pass of Algorithm A, namely  $\mu_1 = \mu_2 = \mu_3 = 2$ . If the increment is cut to 1, then there is no change. Cutting the increment to  $1/2$ , we may raise the middle point to 2.5. Continuing in this fashion, the middle point gets closer and closer to  $2\sqrt{2}$ , the correct result. (Figure 4.4.2 shows the trajectory after passes zero, one, three, and four of Algorithm A'.)

#### 4.4.2. Computational Requirements

The PTIA, unlike dynamic programming, requires relatively little memory; it requires only one floating point number per interpolation point. However, computation of the CPU time requirements is interesting.

Obviously, the computation time must increase at least linearly with the number of interpolation points on the curve. In fact, the time increases as the square of the number of points. To see why this is so, consider what happens when the number of interpolation points is doubled. Since there are twice as many points to check on each pass of the algorithm, the computation time must increase by a factor of two. Recalling that the torque constraints translate into slope constraints, it is

clear that the ratio of the amount by which a  $\mu$ -value may be raised to the distance between  $\lambda$ -values will be approximately constant. Therefore halving the spacing of the interpolation points halves the size of the steps which can be taken in the  $\mu$  direction, thus doubling the number of steps. This factor of two times the factor of two which results directly from doubling the number of points gives a factor of four increase in computation time. If doubling the number of interpolation points quadruples the computation time, then the time dependence is quadratic in the number of points, i.e., the time dependence is of the form  $t = KN^2$ , where  $t$  is the computation time,  $N$  is the number of interpolation points, and  $K$  is a constant.

#### 4.4.3. Numerical Examples

As an example, we again use the Bendix PACS arm. First we consider only constraints on joint torques/forces and motor voltages, without considering constraints on their derivatives. The perturbation trajectory planner was written in the C programming language, and run on a Vax-11/780 under the Unix operating system. The planner was tried with a straight-line path, a geodesic in "inertia space" (see the next chapter), and a joint interpolated path. The traversal times for these paths are 1.79 seconds, 1.59 seconds and 1.80 seconds respectively. Plots of  $\mu$  vs.  $\lambda$ , joint positions vs. time, and motor voltage vs. time are shown in figures 4.4.3a through 4.4.5c. Comparing these results to those obtained in Chapter 5 using the phase plane method shows the traversal times and the various plots to be virtually identical.

To demonstrate the application of the perturbation technique to problems in which there are constraints on the derivatives of the torques, we consider the same

problem with the additional constraint that the time derivatives of the joint torques and forces be less than 100. The time derivatives of the torques are computed using the identity

$$\frac{d u_i}{dt} = \frac{d u_i}{d \lambda} \frac{d \lambda}{dt} = \frac{d u_i}{d \lambda} \mu.$$

The derivative  $\frac{d u_i}{d \lambda}$  was estimated by calculating the difference between the applied torques on successive intervals and dividing by the average of the lengths of the intervals. For a straight-line path with 25 interpolation points, the traversal time is 2.04 seconds. The  $\mu$  vs.  $\lambda$  plot is shown in figure 4.4.6. For 50 points, the traversal time is 2.26 seconds; the phase plane plot is shown in figure 4.4.7. Note that the trajectory has a "bump" in it; the process has not converged to the proper solution. To understand why this happens, consider the situation shown in figure 4.4.8. The solid line shows the current trajectory, and the dashed lines show what happens when either of the two interior points is raised. In either case, a jerk limit is exceeded, even though the jerk constraint would very possibly be met if *both* points were raised simultaneously. Neither point can move before the other does, resulting in a sort of "deadlock". Similar situations can occur with longer sequences of points. If jerk constraints are to be included, then obviously we must prevent this sort of situation from occurring. One means of achieving this goal is to perform the trajectory planning operation several times with some added constraints, relaxing the constraints each time the trajectory is "improved". The constraints used here were simple velocity limits. On each pass, the velocity limit is raised. If the velocity increment is small enough, the top of the phase trajectory remains flat, and the regions of high inflection which cause the anomalies in the phase trajectory never get a chance to

appear. With this modification, a velocity increment of 0.1 at each pass gives the results plotted in figures 4.4.9a through 4.4.9c for a straight line with 50 points. 100 points and a velocity increment of 0.025 gives the results plotted in figures 4.4.10a through 4.4.10c. The calculated traversal times are 2.03 seconds in both cases.

The numerical results obtained using the perturbation trajectory improvement algorithm agree with those obtained by the phase plane method in those cases in which both algorithms can be applied. While the iterative improvement algorithm is not as fast as the phase plane method and not as general as dynamic programming, it is extremely flexible and very easy to program.



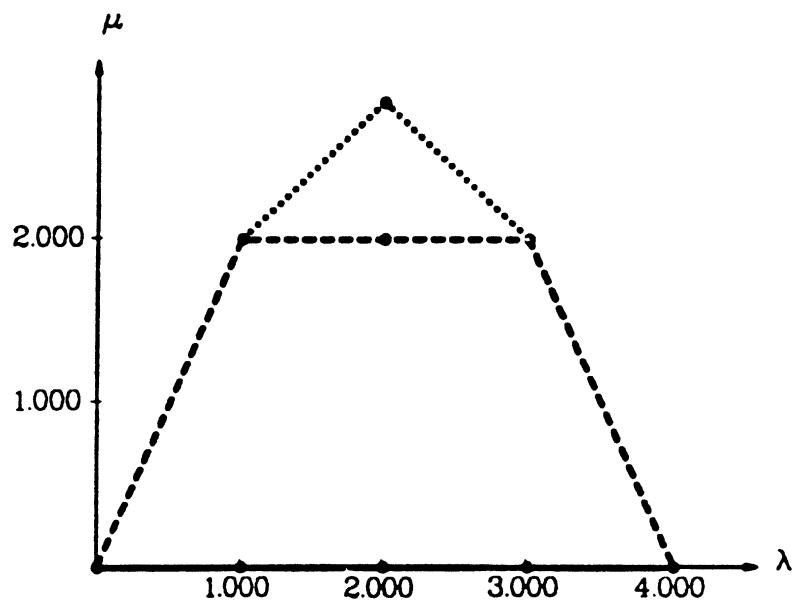


Figure 4.4.1. Results of Algorithm A.

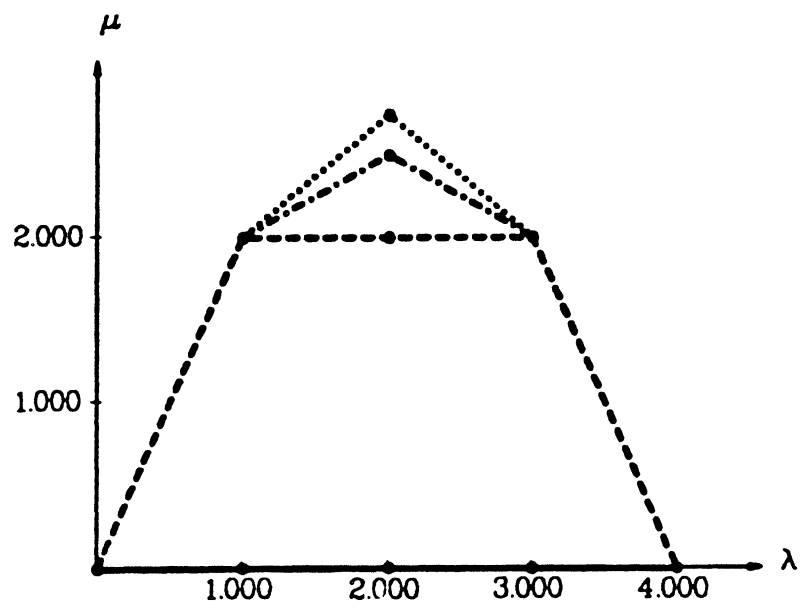


Figure 4.4.2. Results of Algorithm A'.

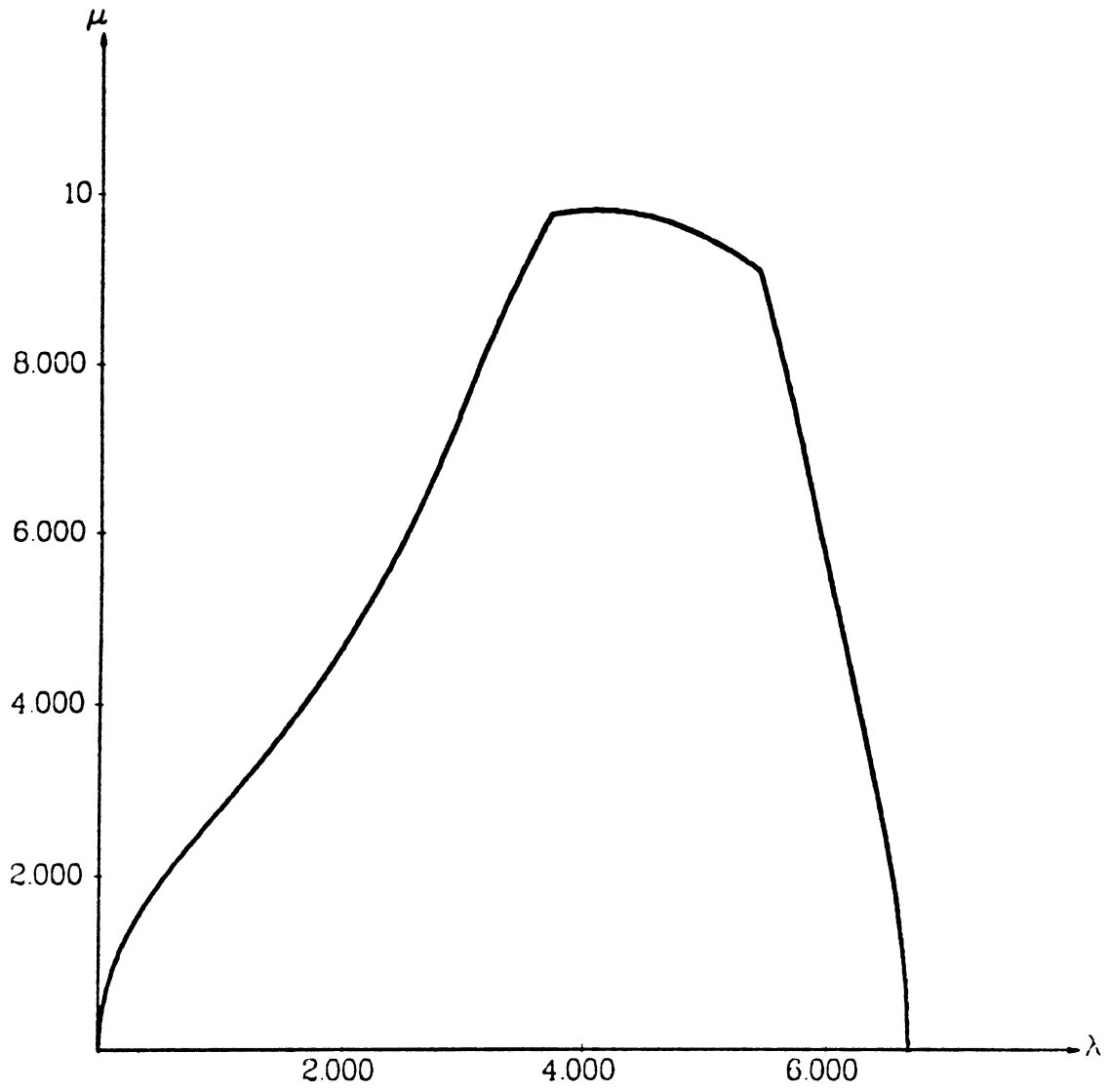


Figure 4.4.3a. Phase plane plot for straight line

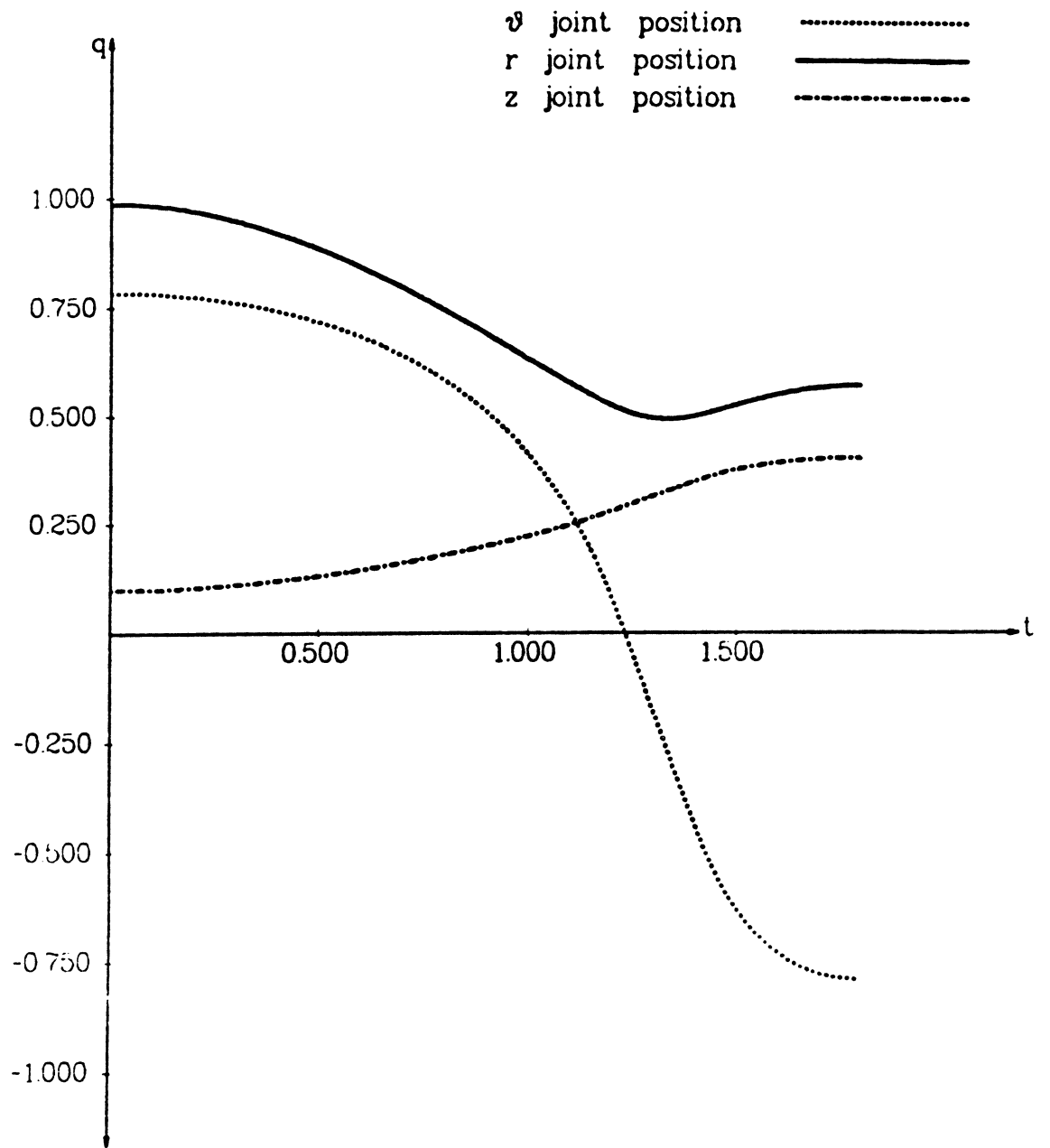


Figure 4.4.3b. Joint position vs. time for straight line

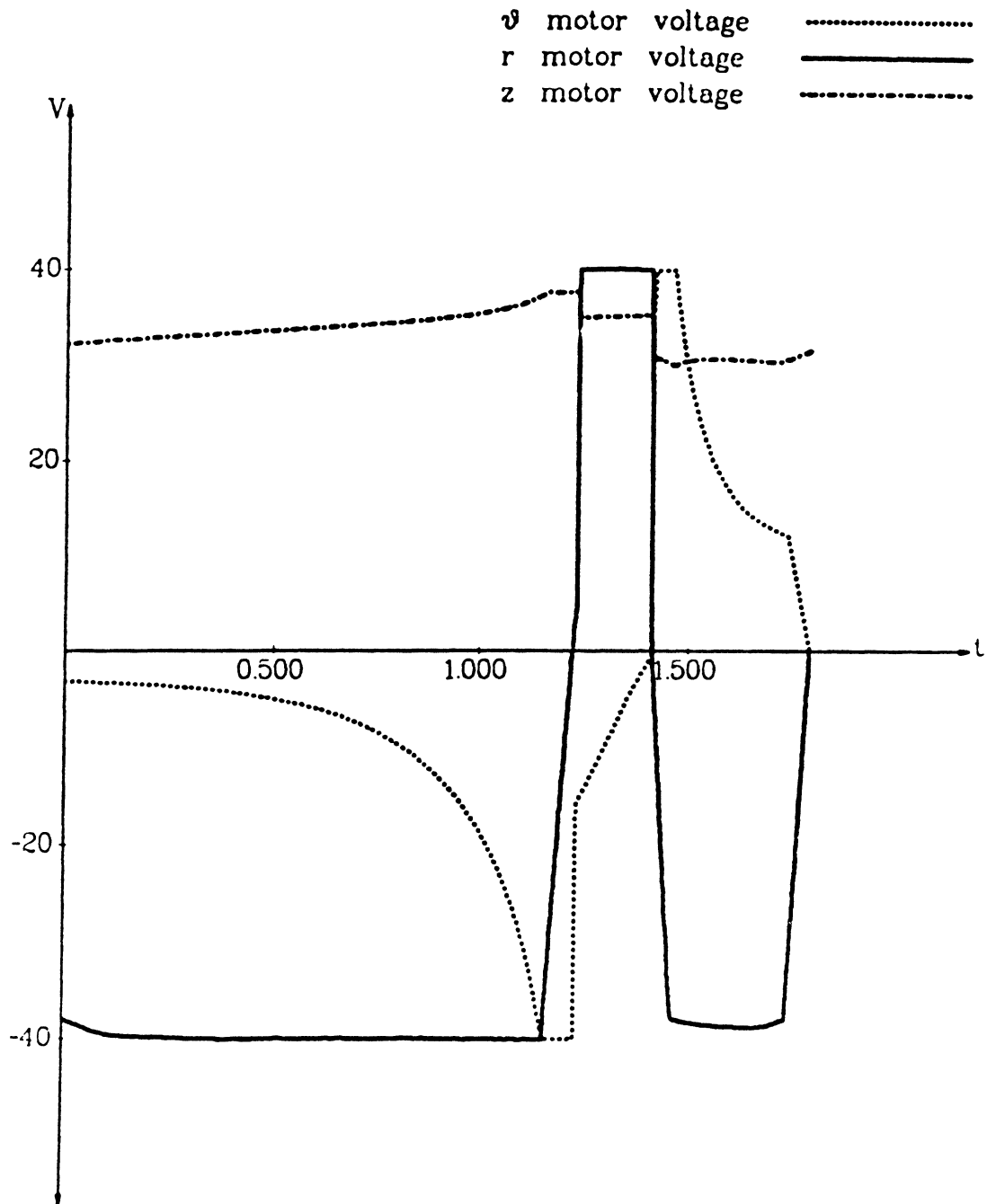


Figure 4.4.3c. Motor voltage vs. time for straight line

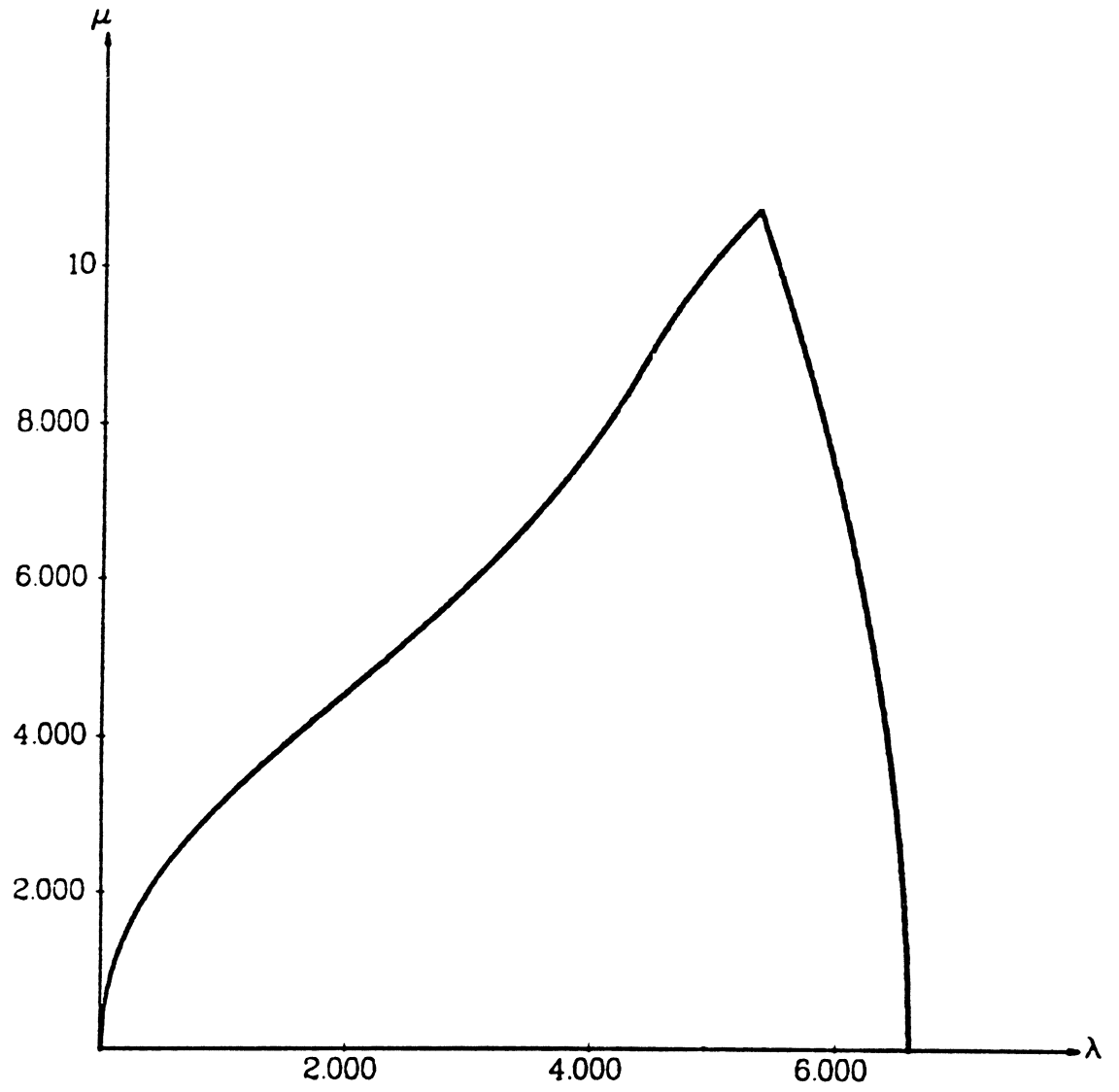


Figure 4.4.4a. Phase plane plot for geodesic

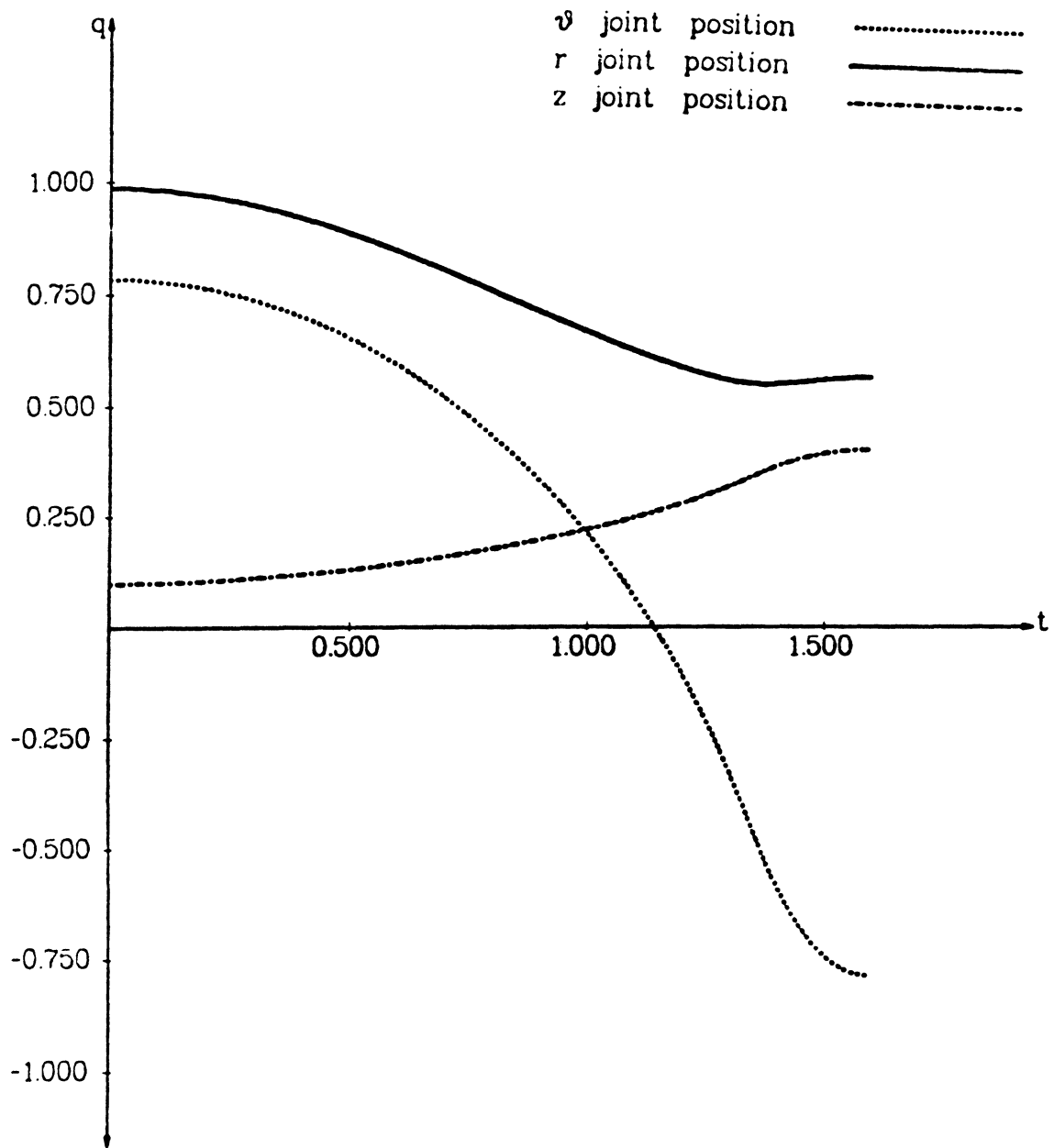


Figure 4.4.4b. Joint position vs. time for geodesic

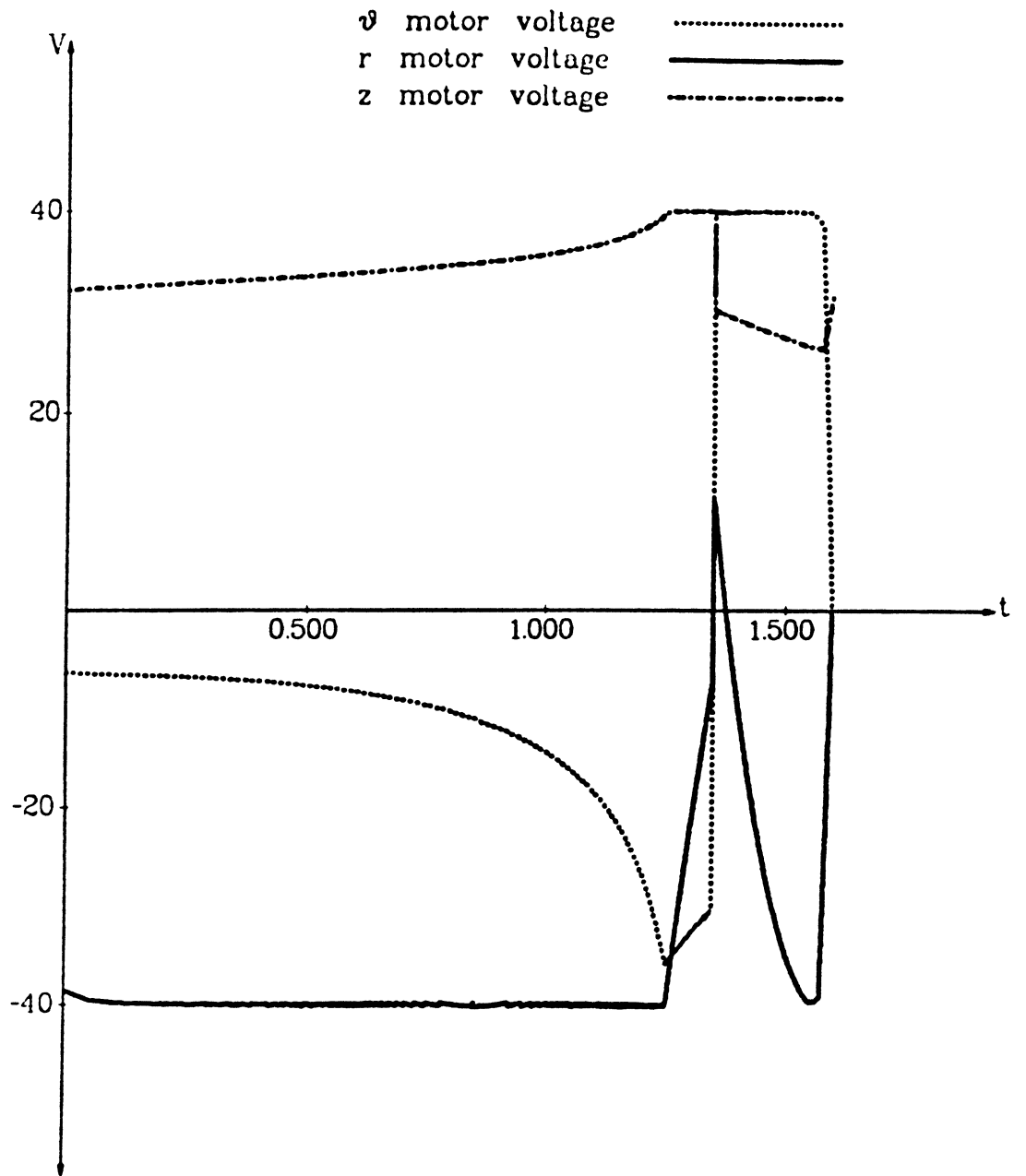


Figure 4.4.4c. Motor voltage vs. time for geodesic



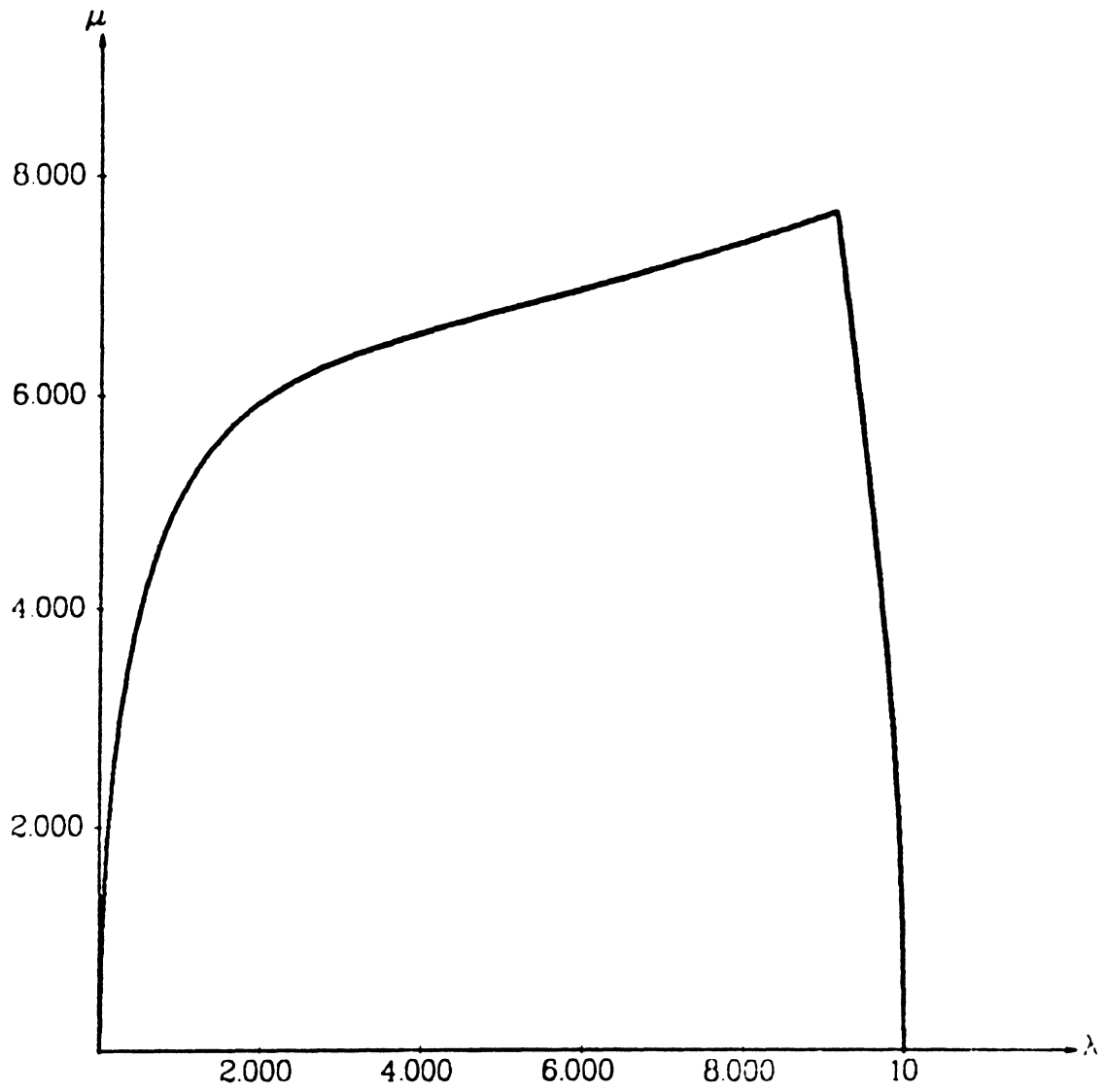


Figure 4.4.5a. Phase plane plot for joint-interpolated curve

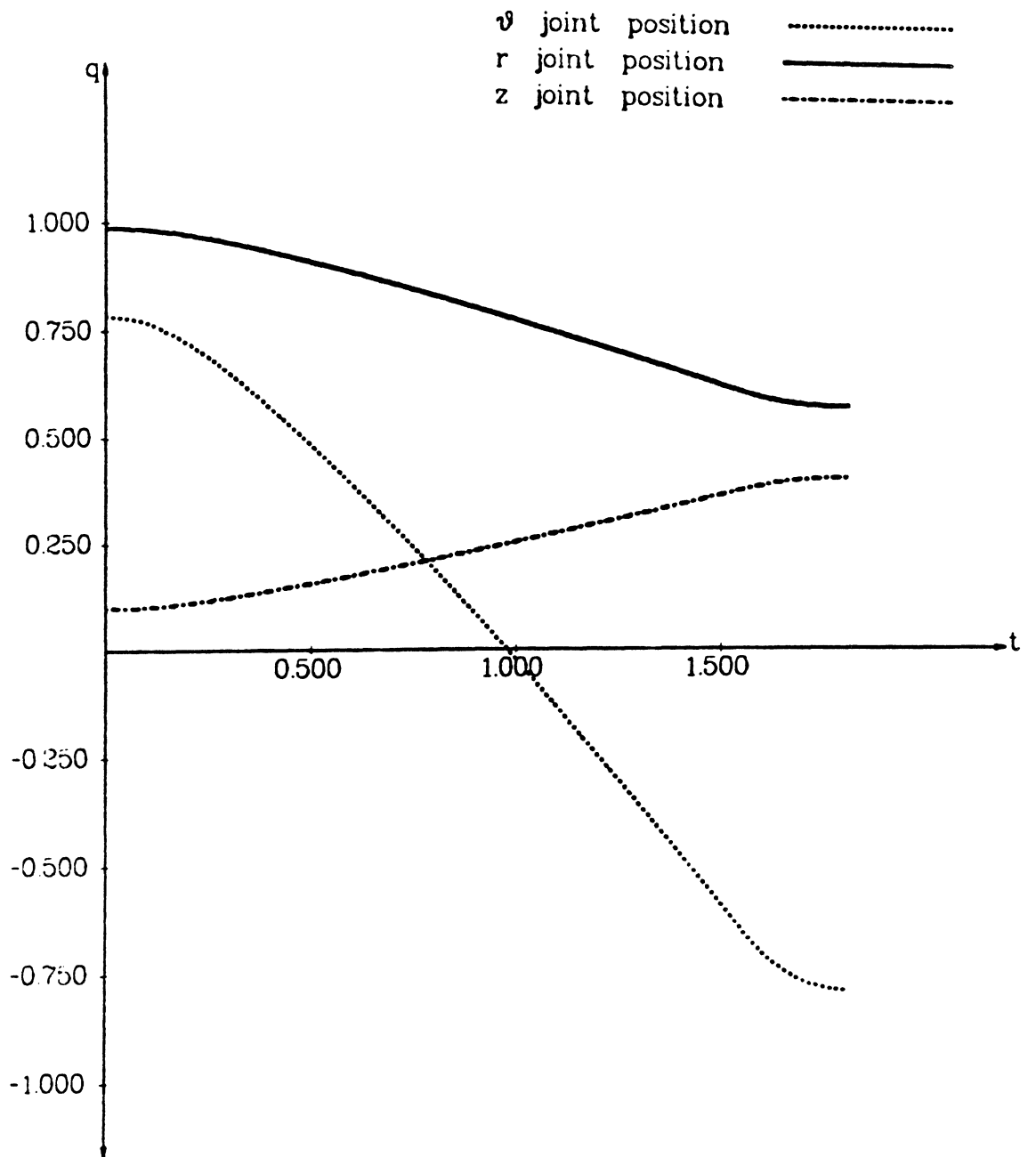


Figure 4.4.5b. Joint position vs. time for joint-interpolated curve

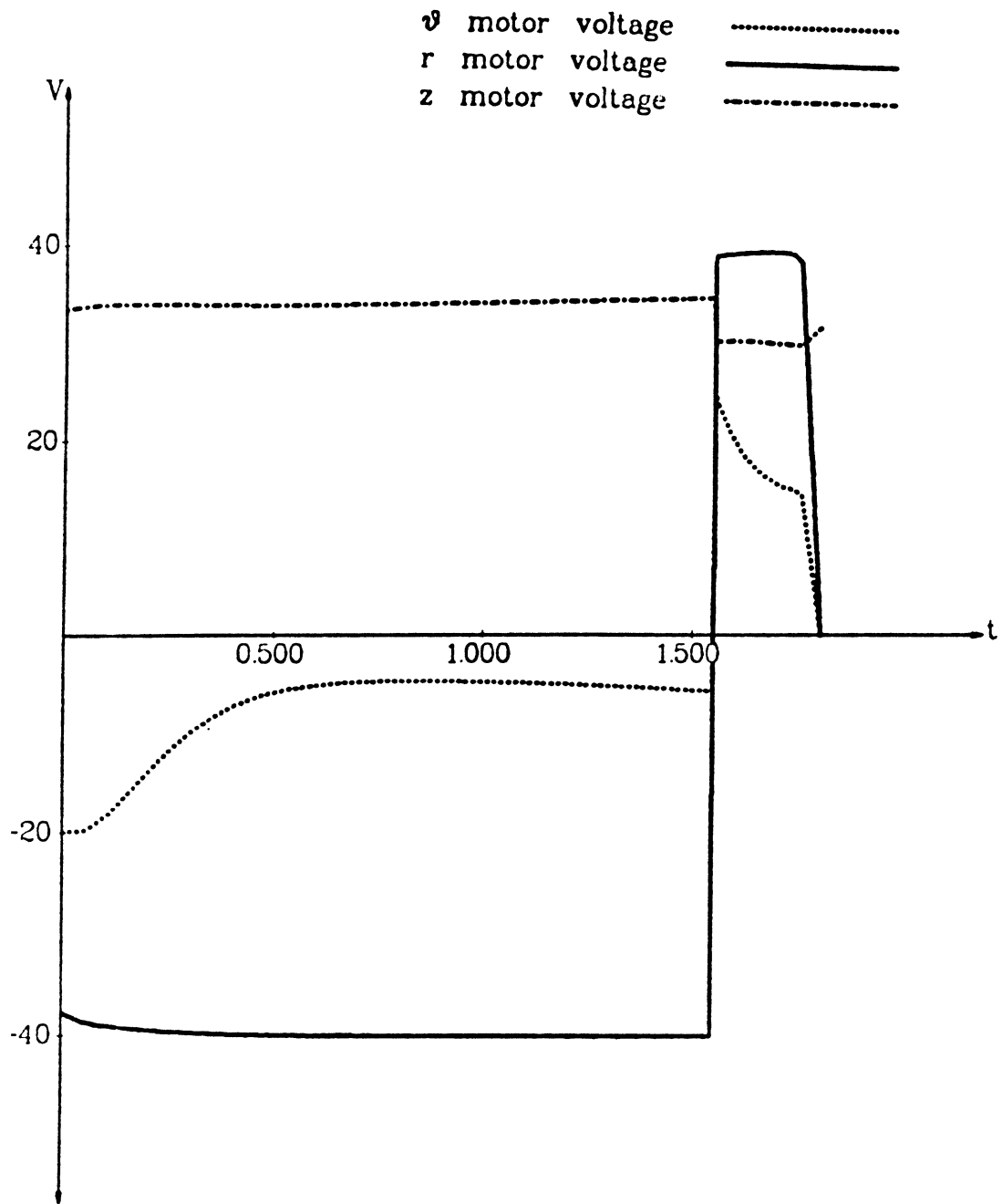


Figure 4.4.5c. Motor voltage vs. time for joint-interpolated curve

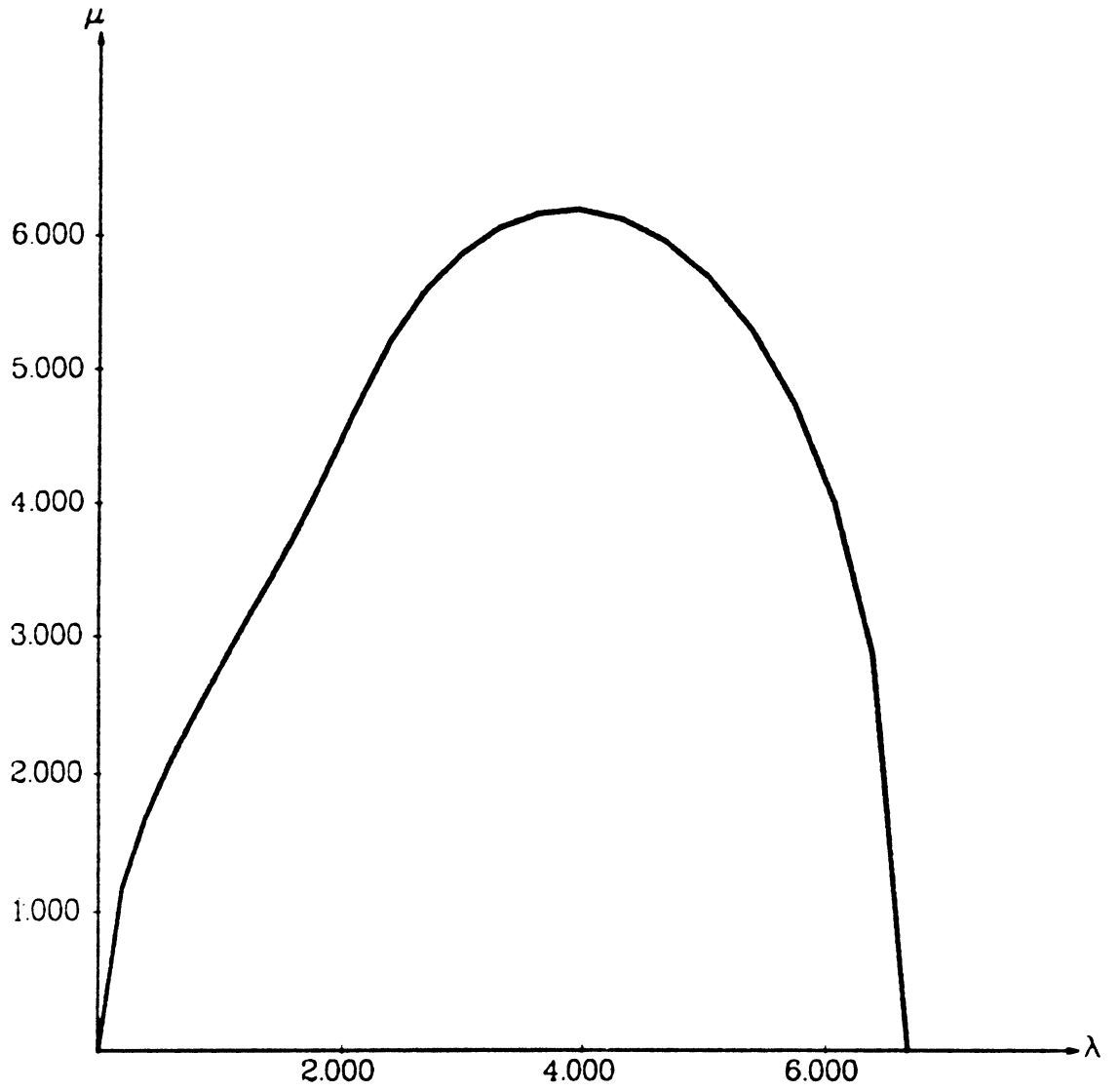


Figure 4.4.6. Phase plane plot for straight line with jerk constraint, 25 points

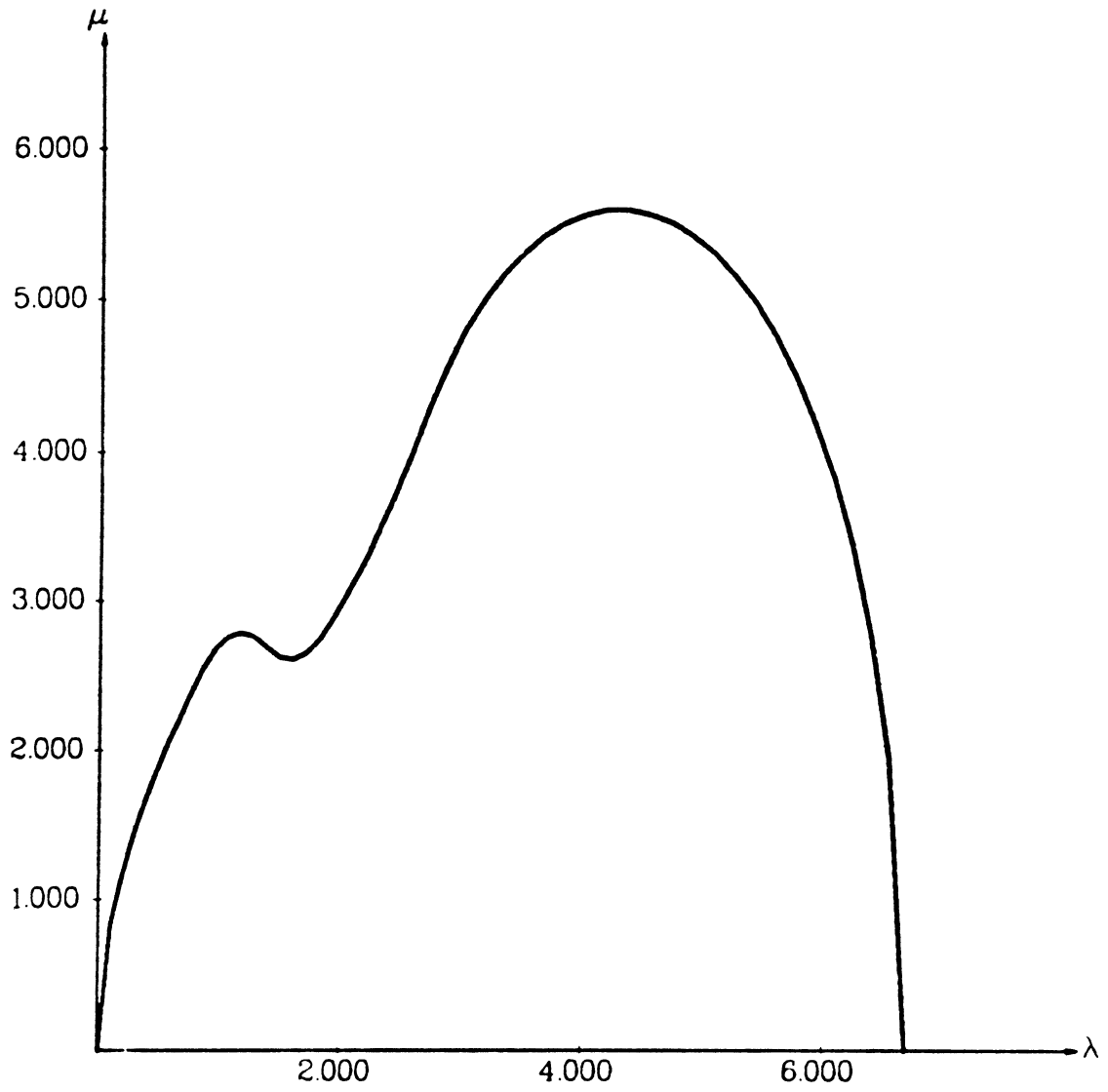


Figure 4.4.7. Phase plane plot for straight line with jerk constraint, 50 points

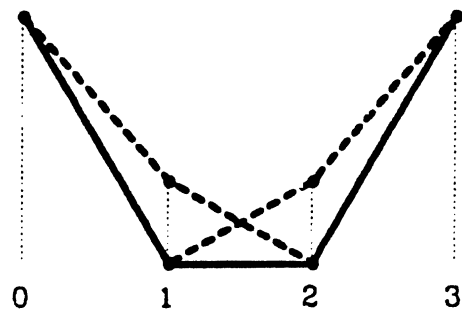


Figure 4.4.8. Illustration of deadlock caused by jerk constraints

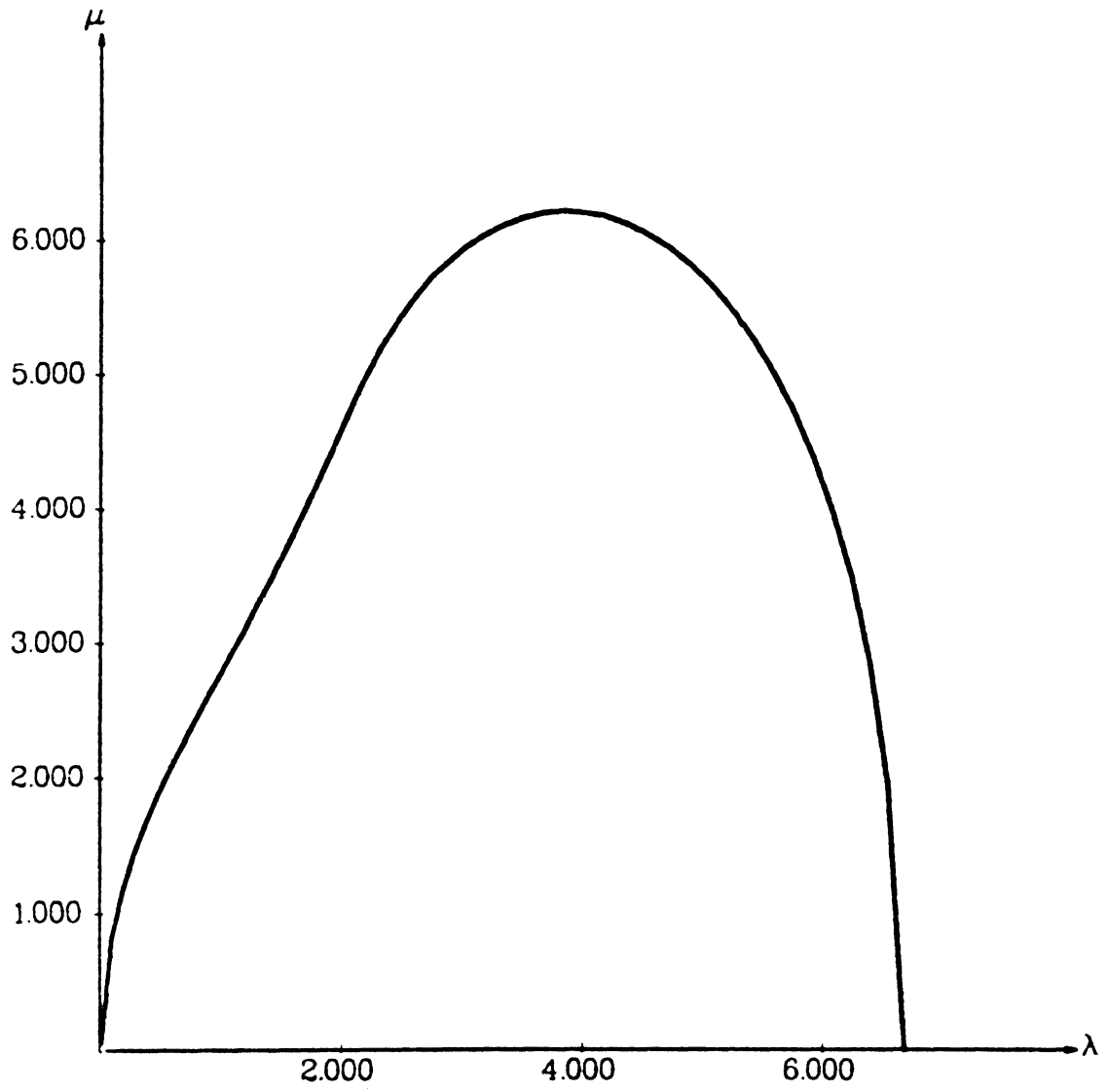


Figure 4.4.9a. Phase plane plot for straight line with jerk constraint, 50 points, velocity increment 0.1





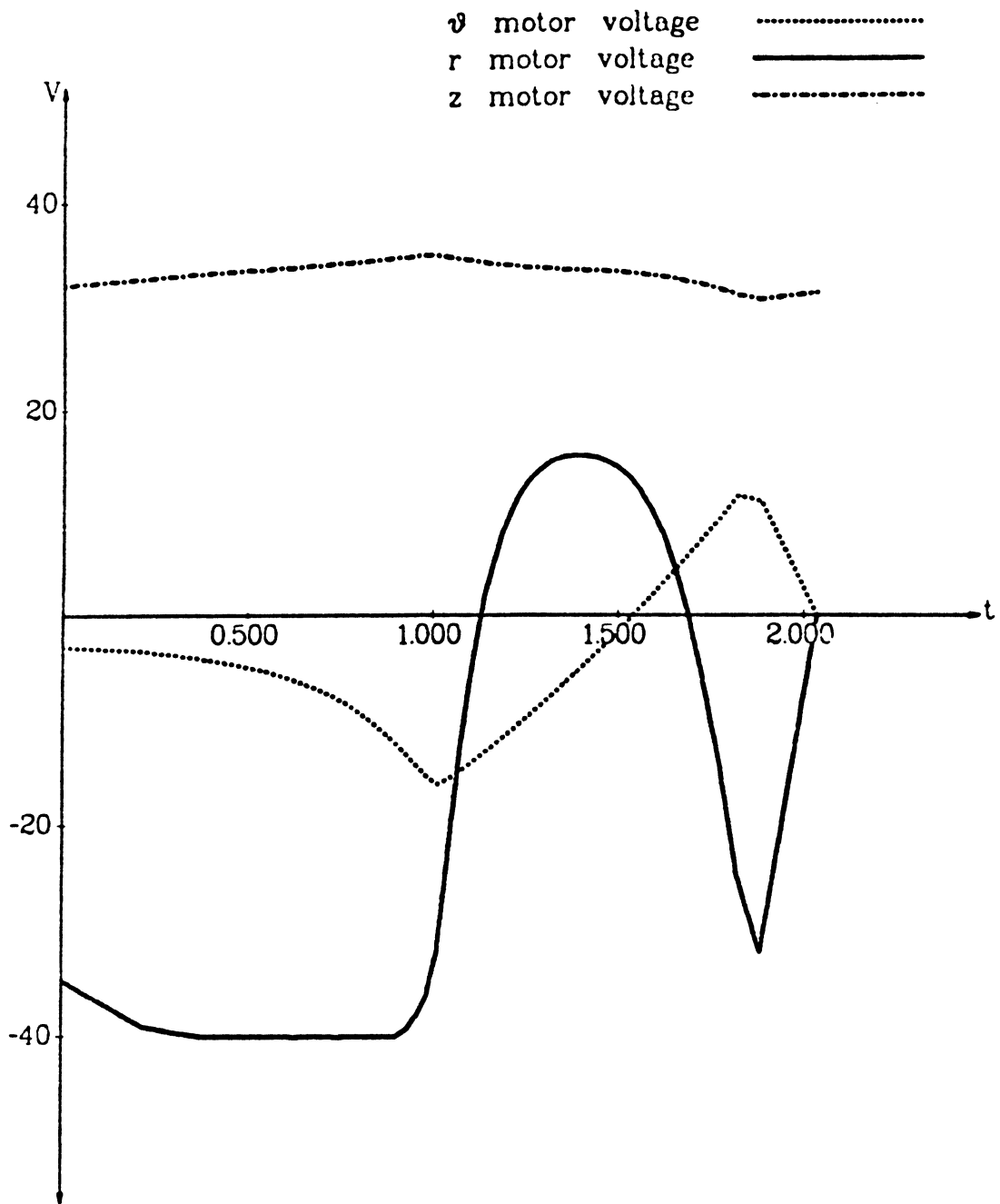


Figure 4.4.9c. Motor voltage vs. time for straight line with jerk constraint, 50 points, velocity increment 0.1

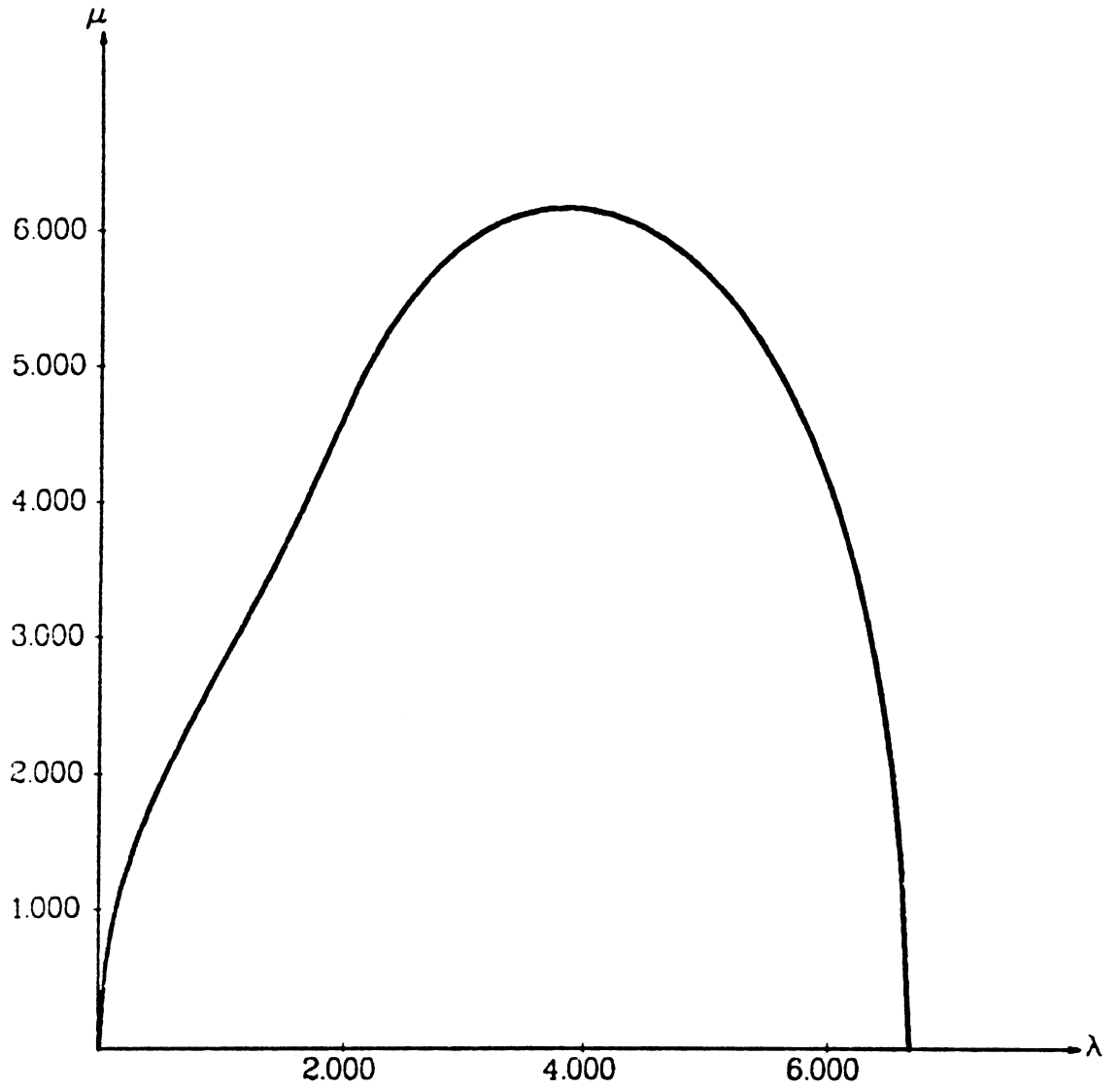


Figure 4.4.10a. Phase plane plot for straight line with jerk constraint, 100 points, velocity increment 0.025

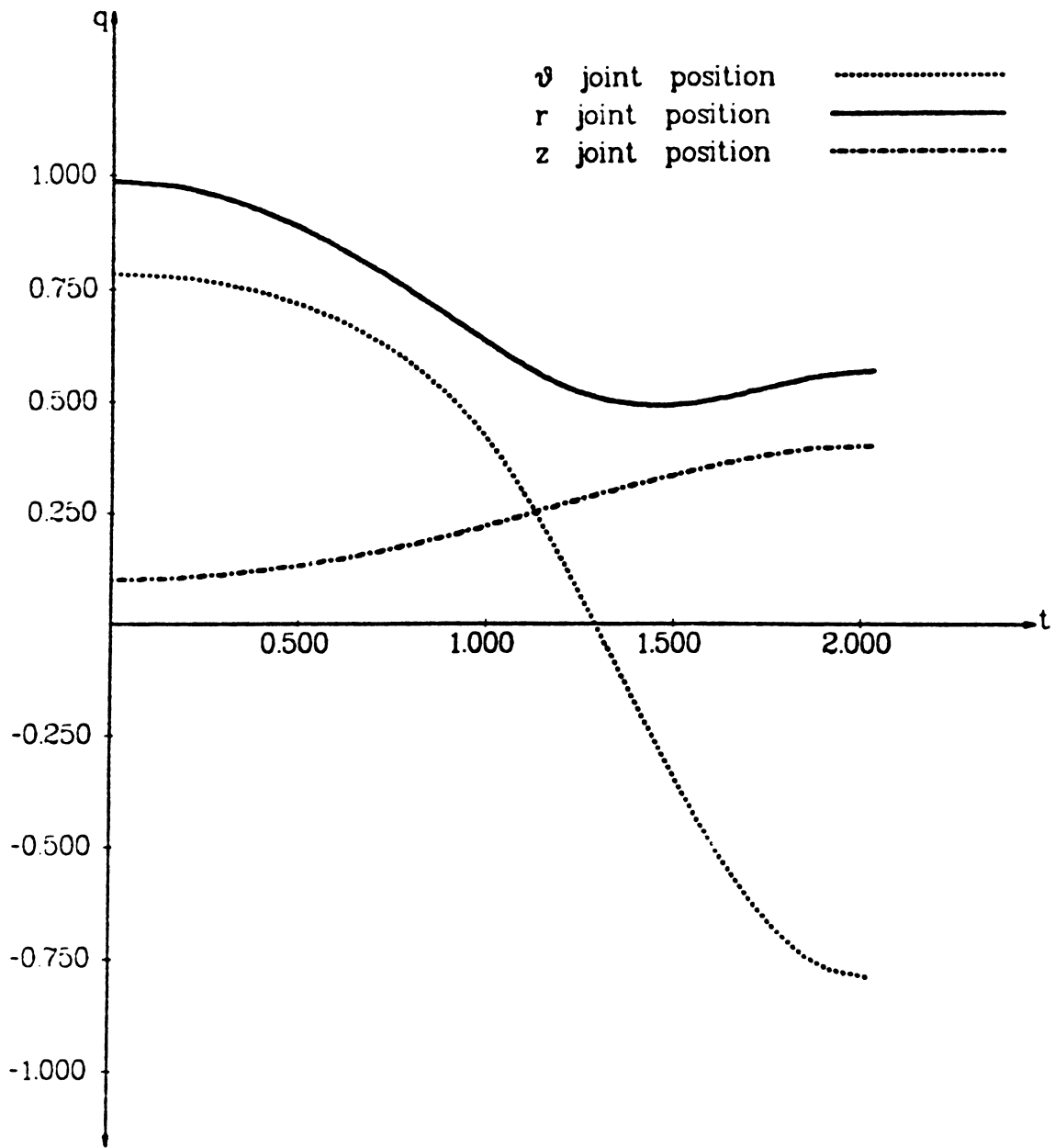


Figure 4.4.10b. Joint position vs. time for straight line with jerk constraint, 100 points, velocity increment 0.025

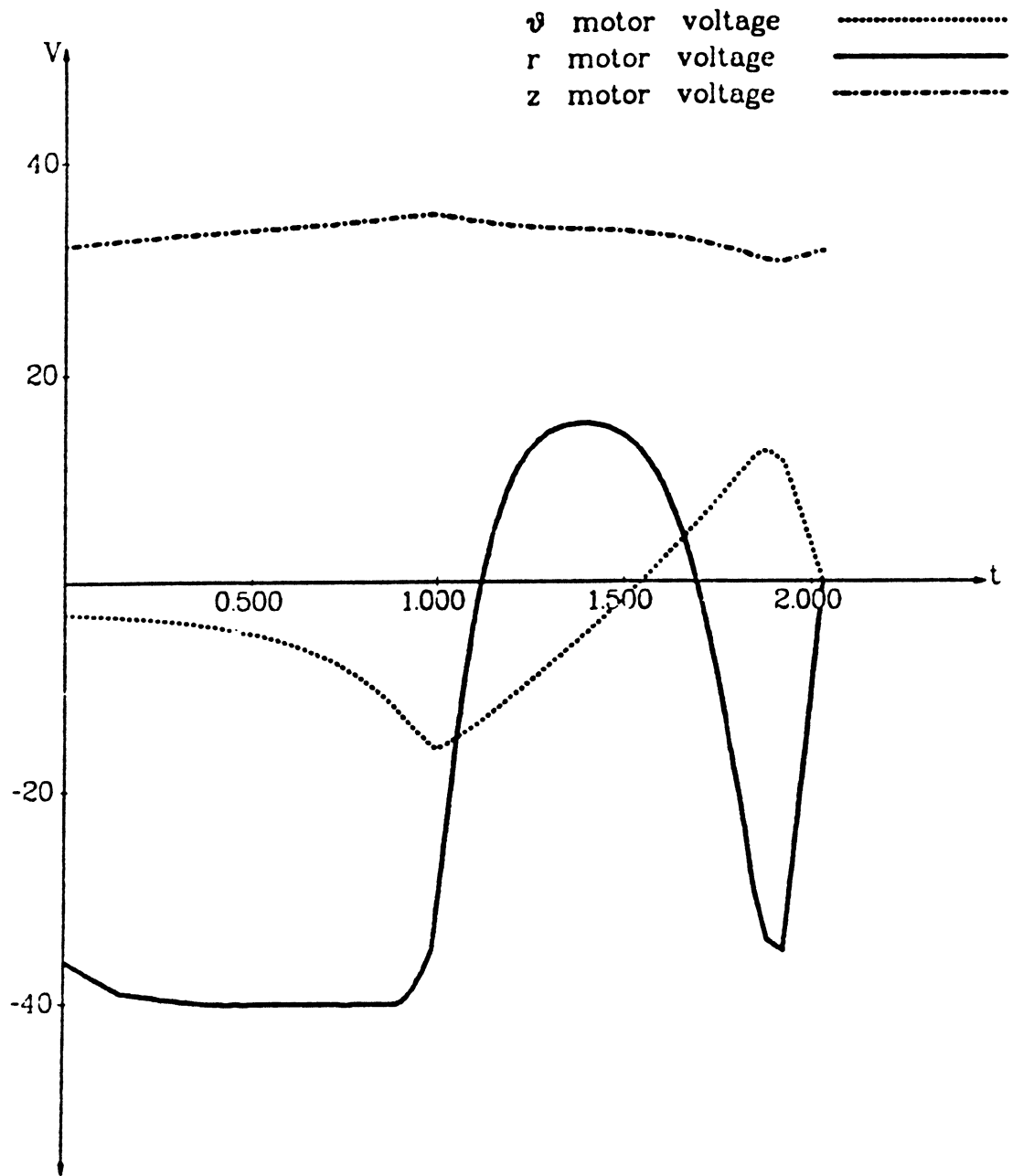


Figure 4.4.10c. Motor voltage vs. time for straight line with jerk constraint, 100 points, velocity increment 0.025

## CHAPTER 5

### SELECTION OF GEOMETRIC PATHS

In general, it is extremely difficult to obtain an exact closed form solution, or even an accurate numerical solution, to the minimum time path planning problem, due to (i) the nonlinearity and coupling in the manipulator dynamics, and (ii) the complexity involved with collision avoidance. The collision avoidance problem (ii) has been sidestepped in the preceding chapters by assuming that the desired geometric or spatial path has been specified *a priori*. It has been assumed that a *task planner* provides a geometric path which is a parameterized curve in joint space. The trajectory planner receives these geometric paths as input and determines a time history of position, velocity, acceleration, and input torques which are then fed to the trajectory tracker. Even if collision avoidance is not a problem, the nonlinearity of the manipulator's dynamics usually makes the minimum-time path planning problem insoluble; it is only by reducing the dimensionality of the problem through the introduction of the parameterized path, i.e., the reduction of the full path planning problem to the trajectory planning problem, that solutions have been obtained in the previous chapter.

In terms of the trajectory planning problem, the geometric path planning problem is the problem of picking the parametric functions  $f^i$  in (4.1.1). In contrast to

the trajectory planning problem, in which the desired solutions can be expressed in terms of the position parameter  $\lambda$  and its first and second time derivatives, the geometric path planning problem requires that a set of functions be chosen from an infinite dimensional space, thereby leading to a more difficult problem.

In this chapter, we will develop a method for determining an approximate minimum time geometric path for the trajectory planners described in chapter 4. It is assumed that the manipulator has an unobstructed workspace, so that collision avoidance is not an issue. While this limits the applicability of the solutions obtained here, it does provide a feeling for what path characteristics determine traversal time, forming a possible base for further work.

### 5.1. Near-Minimum Time Path Planning

As was previously pointed out, use of the maximum principle for solving the minimum-time geometric path planning (MTGPP) problem is practically impossible. Alternative approaches must be sought. In this section we will develop three methods for generating geometric paths. The first two use energy methods to derive a lower bound on path traversal times, and the third method uses the velocity limits derived in Chapter 4.

The energy methods make use of the properties of the "inertia space" described in Chapter 2, especially the formulas (2.4.20) through (2.4.23) relating to power consumption. Using these relations, it will first be shown that geodesics in inertia space, i.e., solutions of the differential equations  $\frac{\delta}{\delta s} \left( \frac{d \mathbf{q}^i}{ds} \right) = 0$ , are the optimal solutions to the MTGPP problem under some restricted conditions. Though the conditions

required in the special case are not met by realistic manipulators, the proof does provide a simple illustration of the method used here for obtaining lower bounds on traversal time; the curves which minimize the lower bound for the more general case can then be found using techniques similar to the one used in the special case, giving an absolute lower bound on the time required to move from one point to another. Then the derived traversal time bounds and the velocity limits derived in Chapter 4 are used to find approximations to minimum time paths.

### 5.1.1. A Special Case

It will now be shown that if a manipulator's dynamic equations have no friction terms and no gravitational terms, and if the limitations on the joint torques consist only of limits on the total power supplied to (or taken from) the manipulator, then the minimum time geometric paths are geodesics in inertia space. Formally, we have the following theorem:

**Theorem 5.1:** If a manipulator is frictionless and has zero gravitational terms, i.e.,  $\mathbf{R}_{ij} = 0$  and  $\mathbf{g}_i = 0$  in the dynamic equations (2.4.12), and the only restrictions on the torques applied to the manipulator arise from constant, symmetric limits on the total power supplied to (or taken from) the manipulator, then the minimum-time geometric path between any two configurations of the manipulator is a geodesic in inertia space provided that the initial and final velocities are zero.

**Proof:** Under the stated conditions the dynamic equations for the manipulator become

$$\mathbf{u}_i = \mathbf{J}_{ij} \dot{\mathbf{v}}^j + [jk, i] \mathbf{v}^j \mathbf{v}^k = \mathbf{J}_{ij} \frac{\delta \mathbf{p}^j}{\delta s} \left( \frac{ds}{dt} \right)^2 + \mathbf{J}_{ij} \mathbf{p}^j \frac{d^2 s}{dt^2}. \quad (5.1.1.1)$$

The total power sunk or sourced by the manipulator is limited by symmetrical constant bounds, i.e.,  $-P_{\max} \leq P \leq P_{\max}$ . Then by Eq. (2.4.23), applying the constant maximum power gives

$$P = P_{\max} = \frac{ds}{dt} \frac{d^2s}{dt^2} = \mu \frac{d\mu}{dt} \quad (5.1.1.2)$$

where  $\mu \equiv \frac{ds}{dt}$ . (Note that the gravitational and friction terms have been dropped from Eq. (2.4.23) for the special case.) Solving the differential Eq. (5.1.1.2) gives

$$\frac{1}{2} \mu^2 = tP_{\max}, \quad \text{or} \quad s = \frac{2}{3} \sqrt{2P_{\max}} t^{\frac{3}{2}} \quad (5.1.1.3)$$

since the manipulator starts at rest.

Obviously, minimizing the traversal time for a given path requires that we maximize the "velocity"  $\frac{ds}{dt}$ . This in turn requires that the power  $P$  be maximized. Therefore, the maximum distance  $s$  which can be traveled in time  $t$  is given by Eq. (5.1.1.3).

Looking now at the end of the curve, we wish to have zero velocity at the end of the motion. Again, since we wish to minimize the traversal time, we want to stop as quickly as possible, which requires that we *drain* energy from the system as fast as possible. Applying the minimum power,

$$\mu \frac{d\mu}{dt} = -P_{\max}. \quad (5.1.1.4)$$

Solving this equation gives



$$\frac{1}{2} \mu^2 = P_{\max}(T - t), \quad \text{or} \quad s = S - \frac{2}{3} \sqrt{2P_{\max}} (T - t)^{\frac{3}{2}}. \quad (5.1.1.5)$$

where  $S$  is the total "length" of the curve which is to be traversed and  $T$  is the (unknown) time when the destination point is reached.

At some point in the middle of the curve there must be a switch from acceleration to deceleration. Let the time, distance, and velocity at this point be denoted by  $t_s$ ,  $s_s$ , and  $\mu_s$ , respectively. Then Eqs. (5.1.1.3) and (5.1.1.5) must give identical results at the switching point, so we have

$$\mu_s^2 = 2P_{\max}t_s = 2P_{\max}(T - t_s) \quad (5.1.1.6)$$

and

$$s_s = S - \frac{2}{3} \sqrt{2P_{\max}} (T - t_s)^{\frac{3}{2}} = \frac{2}{3} \sqrt{2P_{\max}} t_s^{\frac{3}{2}}. \quad (5.1.1.7)$$

If we eliminate  $t_s$  from these equations, we can express  $T$  in terms of  $S$ . The resulting equation is

$$T = \left( \frac{9}{4P_{\max}} \right)^{\frac{1}{3}} S^{\frac{2}{3}} \quad (5.1.1.8)$$

The total time  $T$  increases monotonically with  $S$ , so minimum distance in inertia space is, in this case, equivalent to minimum time. Therefore the geodesic, being the curve of shortest "distance" between any two points, is the optimal geometric path.

■

The conditions under which this proof of optimality applies are not realistic, particularly the condition that gravitational terms be absent. The proof of optimality depends on the absence of gravitational terms because the presence of such terms makes the power supplied to the manipulator a function of position rather than a function only of the kinetic energy of the manipulator. The requirement that the joint torques/forces be only constrained by a total power limit for the entire manipulator is also unrealistic. However, it is possible in practice to obtain bounds on the total available power for the more general case; in the next subsection, such bounds will be used to find bounds on traversal times.

### 5.1.2. Traversal Time Bounds

In this subsection, we show how energy methods similar to those used in the previous subsection may be used to obtain lower bounds on the time required to move from one point in the robot's workspace to another. We start with Eq. (2.4.23), the formula for the power supplied to the manipulator, namely

$$P_t = \frac{ds}{dt} \frac{d^2s}{dt^2} + \mathbf{R}_{ij} \mathbf{p}^i \mathbf{p}^j \left( \frac{ds}{dt} \right)^2 + \mathbf{g}_i \mathbf{p}^i \frac{ds}{dt} \quad (5.1.2.1)$$

where  $P_t$  is the total power supplied to the robot.

If we write  $P_k = \frac{ds}{dt} \frac{d^2s}{dt^2}$ ,  $P_f = \mathbf{R}_{ij} \mathbf{p}^i \mathbf{p}^j \left( \frac{ds}{dt} \right)^2$ , and  $P_g = \mathbf{g}_i \mathbf{p}^i \frac{ds}{dt}$ , then we

have

$$P_k = P_t - P_f - P_g. \quad (5.1.2.2)$$

We will find bounds on  $P_k$  by finding bounds on  $P_t$ ,  $P_f$ , and  $P_g$ .

Before computing these bounds, we need the following result:

**Lemma 5.1:** The 2-norm of the vector  $\mathbf{p}$  is bounded by:

$$\frac{1}{\sqrt{\lambda_{\max}(\mathbf{J})}} = \frac{1}{f_M(\mathbf{J})} \leq \|\mathbf{p}\|_2 \leq \frac{1}{f_m(\mathbf{J})} = \frac{1}{\sqrt{\lambda_{\min}(\mathbf{J})}} \quad (5.1.2.3)$$

where  $f_m(\mathbf{J}) \equiv \min_{\mathbf{p} \neq 0} \sqrt{\frac{\mathbf{p}^T \mathbf{J} \mathbf{p}}{\mathbf{p}^T \mathbf{p}}}$ ,  $f_M(\mathbf{J}) \equiv \max_{\mathbf{p} \neq 0} \sqrt{\frac{\mathbf{p}^T \mathbf{J} \mathbf{p}}{\mathbf{p}^T \mathbf{p}}}$  (the quadratic form  $\mathbf{J}_{ij} \mathbf{p}^i \mathbf{p}^j$  has been written in vector form as  $\mathbf{p}^T \mathbf{J} \mathbf{p}$ ),  $\lambda_{\min}(\mathbf{J})$  is the smallest eigenvalue of  $\mathbf{J}$  for all positions  $\mathbf{q}$ , and  $\lambda_{\max}(\mathbf{J})$  is defined similarly.

**Proof:** Since  $\mathbf{p}^i$  is a unit vector in inertia space, we have  $\mathbf{J}_{ij} \mathbf{p}^i \mathbf{p}^j = \mathbf{p}^T \mathbf{J} \mathbf{p} = 1$ . Then we have

$$1 = \mathbf{p}^T \mathbf{J} \mathbf{p} = \frac{\mathbf{p}^T \mathbf{J} \mathbf{p}}{\mathbf{p}^T \mathbf{p}} \mathbf{p}^T \mathbf{p} \geq \mathbf{p}^T \mathbf{p} \min_{\mathbf{p} \neq 0} \left[ \frac{\mathbf{p}^T \mathbf{J} \mathbf{p}}{\mathbf{p}^T \mathbf{p}} \right] \geq f_m^2(\mathbf{J}) \|\mathbf{p}\|_2^2.$$

Likewise, we have

$$1 = \mathbf{p}^T \mathbf{J} \mathbf{p} = \frac{\mathbf{p}^T \mathbf{J} \mathbf{p}}{\mathbf{p}^T \mathbf{p}} \mathbf{p}^T \mathbf{p} \leq \mathbf{p}^T \mathbf{p} \max_{\mathbf{p} \neq 0} \left[ \frac{\mathbf{p}^T \mathbf{J} \mathbf{p}}{\mathbf{p}^T \mathbf{p}} \right] \leq f_M^2(\mathbf{J}) \|\mathbf{p}\|_2^2.$$

Since  $\mathbf{J}$  is positive definite and symmetric, its eigenvectors span  $R^n$ . Expanding  $\mathbf{p}$  in terms of the eigenvectors of  $\mathbf{J}$  shows that  $f_m^2(\mathbf{J}) = \lambda_{\min}(\mathbf{J})$  and  $f_M^2(\mathbf{J}) = \lambda_{\max}(\mathbf{J})$ , which, combined with the above inequalities, proves the lemma. ■

We now compute bounds on the total applied power  $P_t$ . It will be assumed here that bounds on  $P_t$  arise from constant bounds on the joint torques and from constant bounds on the total applied power. Then, we have

**Lemma 5.2:** If  $\frac{ds}{dt} > 0$ , then  $\max\left\{P_{\min}, -\zeta\frac{ds}{dt}\right\} \leq P_t \leq \min\left\{P_{\max}, \zeta\frac{ds}{dt}\right\}$

where  $P_{\min}$  and  $P_{\max}$  are the minimum and maximum powers that can be supplied to the robot,  $\zeta = \frac{\|\mathbf{u}_i^{\max}\|_2}{\sqrt{\lambda_{\min}(\mathbf{J})}}$ , and  $\|\mathbf{u}_i^{\max}\|_2$  is the maximum 2-norm of the torque vector.

**Proof:** By definition, we have  $P_t = \mathbf{u}_i \mathbf{p}^i \frac{ds}{dt}$ . The component of the torque in the direction of motion,  $\mathbf{u}_i \mathbf{p}^i$ , can be bounded by

$$|\mathbf{u}_i \mathbf{p}^i| \leq \|\mathbf{u}_i^{\max}\|_2 \|\mathbf{p}^i\|_2 \leq \frac{\|\mathbf{u}_i^{\max}\|_2}{\sqrt{\lambda_{\min}(\mathbf{J})}}. \quad (5.1.2.4)$$

We therefore have

$$P_t \geq \max\left\{P_{\min}, \frac{-\|\mathbf{u}_i^{\max}\|_2}{\sqrt{\lambda_{\min}(\mathbf{J})}} \frac{ds}{dt}\right\} = \max\left\{P_{\min}, -\zeta\frac{ds}{dt}\right\} \quad (5.1.2.5)$$

and

$$P_t \leq \min\left\{P_{\max}, \frac{\|\mathbf{u}_i^{\max}\|_2}{\sqrt{\lambda_{\min}(\mathbf{J})}} \frac{ds}{dt}\right\} = \min\left\{P_{\max}, \zeta\frac{ds}{dt}\right\} \quad (5.1.2.6)$$

■

**Lemma 5.3:**  $P_f$  is bounded by  $\phi\mu^2 \leq P_f \leq \phi'\mu^2$  where  $\phi = \frac{\lambda_{\min}(\mathbf{R})}{\lambda_{\max}(\mathbf{J})}$

and  $\phi' = \frac{\lambda_{\max}(\mathbf{R})}{\lambda_{\min}(\mathbf{J})}$ .

**Proof:** By an argument similar to that used to derive bounds on  $\|\mathbf{p}\|_2$ , we have  $\lambda_{\min}(\mathbf{R})\|\mathbf{p}\|_2^2 \leq \mathbf{p}^T \mathbf{R} \mathbf{p} \leq \lambda_{\max}(\mathbf{R})\|\mathbf{p}\|_2^2$  so that, by Lemma 5.1,

$$\frac{\lambda_{\min}(\mathbf{R})}{\lambda_{\max}(\mathbf{J})} \leq \mathbf{p}^T \mathbf{R} \mathbf{p} \leq \frac{\lambda_{\max}(\mathbf{R})}{\lambda_{\min}(\mathbf{J})}. \quad (5.1.2.7)$$

Multiplying by  $\mu^2$  gives the desired result. ■

**Lemma 5.4:** The gravitational energy contribution  $P_g$  is bounded by

$$-\psi \frac{ds}{dt} \leq P_g \leq \psi \frac{ds}{dt} \text{ where } \psi = \frac{\|\mathbf{g}_i\|_2}{\sqrt{\lambda_{\min}(\mathbf{J})}} \frac{ds}{dt}.$$

**Proof:** We have

$$|\mathbf{g}_i \mathbf{p}^i| \leq \|\mathbf{g}_i\|_2 \|\mathbf{p}^i\|_2 \leq \frac{\|\mathbf{g}_i\|_2}{\sqrt{\lambda_{\min}(\mathbf{J})}} \quad (5.1.2.8)$$

by Lemma 5.1. Multiplying by  $\frac{ds}{dt}$  proves the lemma. ■

Using bounds derived in Lemmas 5.2 through 5.4, we are now in a position to obtain bounds on  $P_k$ .

**Lemma 5.5:** If we define  $\mu \equiv \frac{ds}{dt} > 0$ ,

$$\max \left\{ P_{\min}, -\zeta\mu \right\} - \phi'\mu^2 - \psi\mu \leq P_k \leq \min \left\{ P_{\max}, \zeta\mu \right\} - \phi\mu^2 + \psi\mu.$$

Proving this Lemma is just a matter of plugging the bounds obtained in Lemmas 5.2 through 5.4 into Eq. (5.1.2.2).

We can now determine maximum velocities, as was done in the previous subsection. We have the following theorem:

**Theorem 5.2:** If the initial and final velocities are zero, then

$$\mu \leq \min \left\{ \mu_m, \frac{\psi + \zeta}{\phi} (1 - e^{-\phi t}), \frac{\psi + \zeta}{\phi'} (e^{\phi'(T-t)} - 1) \right\}$$

where  $T$  is the traversal time of the path and  $\mu_m = \frac{\psi + \sqrt{\psi^2 + 4\phi P_{\max}}}{2\phi}$ .

**Proof:** Consider two cases. In the first case, let  $P_t$  be limited by the joint torque bounds. Then we have  $-\zeta\mu \leq P_t \leq \zeta\mu$ , so that

$$-\phi'\mu^2 - (\psi + \zeta)\mu \leq \mu \frac{d\mu}{dt} \leq -\phi\mu^2 + (\psi + \zeta)\mu. \quad (5.1.2.9)$$

But then, since we are considering positive values of  $\mu$ ,

$$-\phi'\mu - (\psi + \zeta) \leq \frac{d\mu}{dt} \leq -\phi\mu + (\psi + \zeta). \quad (5.1.2.10)$$

We must have zero velocity at the beginning and end of the path. If the (as yet unknown) traversal time is  $T$ , then

$$\mu \leq \frac{\psi + \zeta}{\phi} (1 - e^{-\phi t}) \quad (5.1.2.11)$$

and

$$\mu \leq \frac{\psi + \zeta}{\phi'} (e^{\phi'(T-t)} - 1). \quad (5.1.2.12)$$

In the second case, limits are imposed by the total power limits, i.e.,  $P_{\min} \leq P_t \leq P_{\max}$ . Then we have

$$P_{\min} - \phi'\mu^2 - \psi\mu \leq \mu \frac{d\mu}{dt} \leq P_{\max} - \phi\mu^2 + \psi\mu. \quad (5.1.2.13)$$

Again, we are only considering positive values of  $\mu$ ; since in general  $P_{\min} < 0$ , the lower bound in this inequality will always be less than zero. Therefore the roots of the lower bound occur for negative values of  $\mu$ , and we cannot place an upper bound on  $\mu$ . On the other hand, the upper bound has a positive root. Since we are starting at  $\mu = 0$ , that value of  $\mu$  for which  $\frac{d\mu}{dt}$  goes to zero cannot be exceeded, and we have

$$\mu \leq \mu_m = \frac{\psi + \sqrt{\psi^2 + 4\phi P_{\max}}}{2\phi}. \quad (5.1.2.14)$$

Since inequalities (5.1.2.11), (5.1.2.12), and (5.1.2.14) must all be met, the theorem follows. ■

To find lower bounds on traversal times, consider a manipulator, call it the *super-manipulator*, for which the constraints on joint torques are such that Eqs. (5.1.2.11), (5.1.2.12) and (5.1.2.14) apply. Then the super-manipulator has limits only on the 2-norm of the tangential component of the torque vector and on the total kinetic energy. Since these constraints apply for the original manipulator, the old manipulator's realizable torques are a subset of the super-manipulator's, so that the super-manipulator can do anything that the original manipulator can do. Thus any path can be traversed by the super-manipulator at least as quickly as the old manipulator could traverse it. Finding the minimum traversal time for the super-manipulator therefore gives a lower bound on the traversal time for the original manipulator.

Finding the lower bound on the traversal time  $T$  for the super-manipulator is simple. It is just a matter of finding a value of  $T$  such that the area under the

velocity vs. time curve is equal to the geodesic distance  $S$  between the initial and final points. Formally, we have

**Theorem 5.3:** Let the times  $t_1$  and  $t_2$  be given by

$$t_1 = \frac{1}{\phi} \log \left[ \frac{\psi + \zeta}{\psi + \zeta - \phi \mu_m} \right] \quad (5.1.2.15)$$

and

$$t_2 = T - \frac{1}{\phi'} \log \left[ \frac{\psi + \zeta + \phi' \mu_m}{\psi + \zeta} \right]. \quad (5.1.2.16)$$

If  $t_1$  and  $t_2$  are both real and  $t_1 \leq t_2$ , then the minimum traversal time  $T$  for the super-manipulator can be found by solving the equation

$$S = \frac{\mu_m}{\phi'} - \frac{\mu_m}{\phi} + \mu_m T - \frac{\mu_m}{\phi'} \log \left[ \frac{\psi + \zeta + \phi' \mu_m}{\psi + \zeta} \right] - \frac{\mu_m}{\phi} \log \left[ \frac{\psi + \zeta}{\psi + \zeta - \phi \mu_m} \right]. \quad (5.1.2.17)$$

If  $t_1$  is not real or  $t_1 > t_2$ , then  $T$  can be found by solving the simultaneous equations

$$\frac{\psi + \zeta}{\phi} (1 - e^{-\phi t_1}) = \frac{\psi + \zeta}{\phi'} (e^{\phi'(T-t_1)} - 1) \quad (5.1.2.18)$$



$$\begin{aligned} & \frac{\psi + \zeta}{\phi} \left[ t_s - \frac{1}{\phi} (1 - e^{-\phi t_s}) \right] \\ & = S - \frac{\psi + \zeta}{\phi'} \left[ \frac{1}{\phi'} (e^{\phi'(T-t_s)} - 1) - (T - t_s) \right]. \end{aligned} \quad (5.1.2.19)$$

**Proof:** Finding the distance travelled, the area under the velocity vs. time curve, requires that we consider the two cases described above, which correspond to (i) the case in which the velocity limit  $\mu_m$  is reached, and (ii) the case where it is not. Case (i) is relatively simple. First, we need to know the points where the curves described in (5.1.2.11) and (5.1.2.12) reach the limiting velocity  $\mu_m$ . These times may be obtained from (5.1.2.11) and (5.1.2.12) by setting  $\mu = \mu_m$  and solving for  $t$ . These times are just  $t_1$  for (5.1.2.11) and  $t_2$  for (5.1.2.12). Then the area under the velocity vs. time curve will be given by

$$\begin{aligned} S &= \int_0^{t_1} \frac{\psi + \zeta}{\phi} (1 - e^{-\phi t}) dt + \int_{t_1}^{t_2} \mu_m dt + \int_{t_2}^T \frac{\psi + \zeta}{\phi'} (e^{\phi'(T-t)} - 1) dt \\ &= \frac{\mu_m}{\phi'} - \frac{\mu_m}{\phi} + \mu_m T - \frac{\mu_m}{\phi'} \log \left[ \frac{\psi + \zeta + \phi' \mu_m}{\psi + \zeta} \right] \\ &\quad - \frac{\mu_m}{\phi} \log \left[ \frac{\psi + \zeta}{\psi + \zeta - \phi \mu_m} \right]. \end{aligned} \quad (5.1.2.20)$$

Since  $S$  is linear in  $T$ , determining  $T$  is easy.

In case (ii), we have a single switching time  $t_s$ . We may match positions and velocities as was done in the special case in the previous subsection, giving the

equations

$$\mu_s = \frac{\psi + S}{\phi} (1 - e^{-\phi t_s}) = \frac{\psi + S}{\phi'} (e^{\phi(T-t_s)} - 1) \quad (5.1.2.21)$$

$$s_s = \int_0^{t_s} \frac{\psi + S}{\phi} (1 - e^{-\phi t}) dt = S - \int_{t_s}^T \frac{\psi + S}{\phi'} (e^{\phi(T-t)} - 1) dt \quad (5.1.2.22)$$

$$= \frac{\psi + S}{\phi} \left[ t_s - \frac{1}{\phi} (1 - e^{-\phi t_s}) \right] = S - \frac{\psi + S}{\phi'} \left[ \frac{1}{\phi'} (e^{\phi(T-t_s)} - 1) - (T - t_s) \right].$$

These are just the equations given in the statement of the theorem. ■

Unfortunately, Eqs. (5.1.2.21) and (5.1.2.22) cannot be solved for  $t_s$  in closed form. However, we can still use these equations to prove that  $T$  increases monotonically with  $S$ , and thus prove that the optimal path for the super-manipulator is a geodesic. This being known, the geodesic distance  $S$  between the initial and final points can be calculated, and Eq. (5.1.2.22) can be solved numerically.

**Theorem 5.4:** The minimum traversal time  $T$  for the super-manipulator increases monotonically with the geodesic length  $S$  of the traversed path.

**Proof:** To prove that  $T$  increases monotonically with  $S$ , we will show that  $\frac{dT}{dS} > 0$ . If case (i) of Theorem 5.3 holds, then the result is obvious. Case (ii) is slightly more complicated. First, we differentiate (5.1.2.21) and (5.1.2.22) with respect to the switching time  $t_s$ , giving

$$e^{-\phi t_s} = e^{\phi(T-t_s)} \left( \frac{dT}{dt_s} - 1 \right) \quad (5.1.2.23)$$

$$\frac{\psi + \zeta}{\phi} (1 - e^{-\phi t_s}) = \frac{dS}{dt_s} - \frac{\psi + \zeta}{\phi'} \left( \frac{dT}{dt_s} - 1 \right) (e^{-\phi(T-t_s)} - 1). \quad (5.1.2.24)$$

Solving (5.1.2.23) for  $\frac{dT}{dt_s}$ , plugging into (5.1.2.24), solving (5.1.2.24) for  $\frac{dS}{dt_s}$ , and dividing  $\frac{dS}{dt_s}$  by  $\frac{dT}{dt_s}$  gives

$$\frac{dS}{dT} = \frac{\psi + \zeta}{\phi \phi' (1 + e^{-\phi t_s} e^{-\phi(T-t_s)})} \left[ \phi' (1 - e^{-\phi t_s}) + \phi e^{-\phi t_s} (1 - e^{-\phi(T-t_s)}) \right] \quad (5.1.2.25)$$

which is greater than zero. ■

If the actual lower bound is required, then Eq. (5.1.2.21) may be solved for  $T$ . To do this, we may make use of the equations for  $t_1$  and  $t_2$ . The maximum velocity  $\mu_m$  can be varied in these equations until  $t_1 = t_2 = t_s$ ; then this value of  $t_s$  can be used in (5.1.2.23), which can be solved for  $T$ .

### 5.1.3. Approximate Minimum Time Paths

In this subsection we consider two methods for generating geometric paths which are approximately minimum time. The first method uses the traversal time bounds derived in the previous section and the second method uses the velocity bounds derived in Chapter 4.

First, consider the lower bounds on traversal time. If these bounds are good estimates of the actual traversal times, then minimizing the lower bound should approximately minimize the traversal time. Since the lower bound increases monotonically with the geodesic length  $S$  of the traversed curve, geodesics (minimum-length curves) must minimize the lower bound. The "near optimal" paths may then be determined by solving the differential equations for a geodesic, namely  $\frac{\delta}{\delta s} \left( \frac{d q^i}{d s} \right) = 0$ . This method of generating near-minimum time paths can be applied to most practical robots; however, it places no penalties on forces which are orthogonal to the traversed path, so that path curvature is not penalized. If any further constraints are applied which force the introduction of curvature terms, then ignoring the magnitude of the curvature terms could make the lower bound a poor estimate of the actual traversal time, causing a poor choice of path. The minimization of lower bounds leads to the selection of shortest-distance paths in inertia space, which could have corners at which the manipulator must come to a complete stop. Path segments of high curvature also slow the manipulator down. Thus it is necessary to strike a compromise between curves of shortest distance and curves of smallest curvature. (This naturally leads to the second method of generating near-minimum time geometric paths.)

In order to reach such a compromise, we choose as an objective function the product of the length of the curve and some measure of the total curvature. This, of course, requires some quantitative measure of both curvature and distance in an  $n$ -dimensional space where  $n$  is the number of manipulator joints. One obvious measure of total curvature is the reciprocal of the maximum velocity, as computed in Chapter 4. If the path is expressed in terms of an arbitrary parameter  $\lambda$ , then the

expression

$$\int_0^{\lambda_{\max}} \frac{d\lambda}{\mu_{\max}(\lambda)} \quad (5.1.3.1)$$

would appear to be a good choice, where  $\mu \equiv \dot{\lambda}$  and  $\mu_{\max}(\lambda)$  is the velocity limit at position  $\lambda$ . This expression is independent of the parameterization chosen, and increases both as the length of the curve increases and as the curvature increases.

In order to use (5.1.3.1), the value of the maximum velocity  $\mu_{\max}(\lambda)$  needs to be computed. This is calculated from equation (4.2.1.1), namely

$$u_i^{\min} \leq M_i \dot{\mu} + Q_i \mu^2 + R_i \mu + S_i \leq u_i^{\max}. \quad (5.1.3.2)$$

For a given position  $\lambda$  and velocity  $\mu$ , these inequalities give a range of accelerations  $\dot{\mu}$ , and so may be thought of as assigning upper and lower acceleration bounds to each point  $(\lambda, \mu)$  in the phase plane. Since these inequalities must hold for all joints of the manipulator, the acceleration must fall between the greatest of the lower acceleration bounds and the least of the upper bounds. When one of the upper acceleration bounds is smaller than one of the lower acceleration bounds for some phase point  $(\lambda, \mu)$ , there are no accelerations which will keep the manipulator on the desired path. Thus the acceleration bounds generate restrictions on the velocities at the phase points which can be encountered during a traversal of the path. These relationships can be thought of as assigning velocity limits to a given position  $\lambda$ .

Now consider a frictionless manipulator, i.e., one for which the quantities  $R_i$  are zero. Also assume that at every point on the path the manipulator is capable of stopping and holding its position. Then we have

$$\mathbf{u}_i^{\min} \leq S_i \leq \mathbf{u}_i^{\max} \quad (5.1.3.3)$$

at all points on the path. (This will hereafter be referred to as the “strong manipulator assumption”). If the parameter  $\lambda$  is defined to be the arc length  $s$  in inertia space, then  $Q_i$  is just the inertia matrix  $J_{ij}$  multiplied by the curvature vector  $\frac{\delta \mathbf{p}^j}{\delta s}$ .

If the path chosen is a geodesic, then the curvature vector is zero, and hence  $Q_i \equiv 0$ . Then the inequality (5.1.3.2) reduces to

$$\mathbf{u}_i^{\min} \leq M_i \dot{\mu} + S_i \leq \mathbf{u}_i^{\max} \quad (5.1.3.4)$$

which is independent of the velocity  $\mu$ , and by the strong manipulator assumption is satisfied identically for  $\dot{\mu}=0$ . But if the bounds on  $\dot{\mu}$  are independent of  $\mu$ , there can be no velocity limits; in other words,  $\mu_{\max}(\lambda) = \infty$ , so that the integrand of (5.1.3.1) is zero. Thus in this case the optimal solution coincides with that obtained from minimizing traversal time bounds.

It may appear at first that the geodesic, since it maximizes velocity bounds, must be the true minimum time path. However, the manipulator must meet acceleration as well as velocity constraints. It would then be expected that along the optimal geometric path the maximum acceleration would be maximized during an accelerating portion of the path and the minimum acceleration would be minimized during a decelerating portion. This does not happen along a geodesic, but a similar phenomenon occurs: the acceleration bounds “spread out”. To see this, note that velocity limits occur because the acceleration bounds become very close. Since the velocity bounds have been eliminated by choosing a geodesic as the path, the

acceleration bounds must never get close. Hence maximizing velocity bounds also gives a large range of accelerations to choose from. This would lead one to expect that geodesics are good, if not optimal, choices for geometric paths.

In summary, we have two criteria for selecting near-minimum time geometric paths for a manipulator, one based on the minimization of a lower bound on the manipulator's traversal time and the other based on the minimization of the product of the path's length and its curvature. In either case, when the near-optimal path is determined, the path is found to be a geodesic in inertia space. This geodesic can be constructed by solving a set of differential equations and applying appropriate boundary conditions.

One comment on calculating the geodesics is in order. Finding the geodesic which passes through two points is a two-point boundary value problem. Solving the problem using Pontryagin's maximum principle also gives rise to a two-point boundary value problem, so one may legitimately ask what the advantage of using geodesics is. The important difference is that the boundary conditions for finding the geodesic are easier to apply than are the boundary conditions which arise when using the maximum principle. The geodesics may be found using shooting methods, which involve guessing the initial direction of the geodesic, and refining the guess until the curve comes sufficiently close to the desired endpoint.

## 5.2. Numerical Examples

To demonstrate the utility of the solutions described above, the traversal times for various geometric path have been calculated, using the phase plane method, for the first three joints of the Bendix PACS arm. We construct three paths: a straight

line, a geodesic, and a joint-interpolated curve. (The joint-interpolated curve has the form  $\mathbf{q}^i = \mathbf{q}_s^i + \lambda(\mathbf{q}_f^i - \mathbf{q}_s^i)$ , where  $0 \leq \lambda \leq 1$  and  $\mathbf{q}_s^i$  and  $\mathbf{q}_f^i$  are the points at which the curve starts and finishes.)

The construction of geodesics requires that the inertia matrix of the robot,  $\mathbf{J}_{ij}$ , be known. For the first three joints of the manipulator the inertia matrix takes the form

$$\mathbf{J}_{ij} = \begin{bmatrix} J_t - Kr + M_t r^2 & 0 & 0 \\ 0 & M_t & 0 \\ 0 & 0 & M_z \end{bmatrix} \quad (5.2.1)$$

where  $q^1 = \theta$ ,  $q^2 = r$ , and  $q^3 = z$ . The constants  $M_t$  and  $M_z$  are the masses which the  $r$  and  $z$  axes must move.  $J_t$  is the moment of inertia around the  $\theta$  axis when  $r$  is zero. The  $K$  term is present because the center of mass of the structure for the  $r$  joint does not coincide with the  $\theta$  axis when  $r$  is zero. The form of the matrix is derived in the Appendix. The values of  $J_t$ ,  $K$ ,  $M_t$ , and  $M_z$ , along with friction coefficients and actuator characteristics, may also be found there. The non-zero Christoffel symbols of the first kind (Coriolis coefficients), found by differentiating  $\mathbf{J}_{ij}$ , are

$$[12,1] = [21,1] = M_t r - \frac{K}{2} \quad (5.2.2)$$

$$[11,2] = \frac{K}{2} - M_t r. \quad (5.2.3)$$

The geodesics are solutions of the equations  $0 = \mathbf{J}_{ij} \frac{d^2 \mathbf{q}^j}{ds^2} + [jk, i] \frac{d \mathbf{q}^j}{ds} \frac{d \mathbf{q}^k}{ds}$ .

Plugging Eqs. (5.2.1) through (5.2.3) into this equation gives the equations of the



geodesics as

$$0 = (J_t - Kr + M_t r^2) \frac{d^2 \theta}{ds^2} + (2M_t r - K) \frac{dr}{ds} \frac{d\theta}{ds} \quad (5.2.4)$$

$$0 = M_t \frac{d^2 r}{ds^2} + \left( \frac{K}{2} - M_t r \right) \left( \frac{d\theta}{ds} \right)^2 \quad (5.2.5)$$

$$0 = M_z \frac{d^2 z}{ds^2}. \quad (5.2.6)$$

In addition, we have the normality condition

$$(J_t - Kr + M_t r^2) \left( \frac{d\theta}{ds} \right)^2 + M_t \left( \frac{dr}{ds} \right)^2 + M_z \left( \frac{dz}{ds} \right)^2 = 1. \quad (5.2.7)$$

The differential equations (5.2.4), (5.2.5), and (5.2.6) can be solved in terms of elliptic integrals. In practice, however, it is simpler to solve them numerically.

The gravitational terms for this manipulator are particularly simple; the gravitational forces on the  $r$  and  $\theta$  joints are zero, and the force on the  $z$  joint is  $M_z g$ .

A phase plane trajectory planner for the PACS robot was written in the C programming language and run under the UNIX<sup>3</sup> operating system on a VAX-11/780<sup>4</sup>. The trajectory planner was used to generate trajectories for a straight line, a geodesic, and a joint interpolated curve, each of which extended from the Cartesian point (0.7,0.7,0.1) to (0.4,-0.4,0.4), all coordinates being measured in meters. Phase plane plots (plots of the speed  $\mu$  versus position  $\lambda$ ), plots of position  $q^i$  vs. time, and plots of motor voltage vs. time are shown in Figures 5.2.1a through 5.2.3c. Figures

---

<sup>3</sup>UNIX is a trademark of Bell Laboratories.

<sup>4</sup>VAX is a trademark of Digital Equipment Corporation.

5.2.1a through 5.2.1c are for the straight line, Figures 5.2.2a through 5.2.2c are for the joint interpolated curve and Figures 5.2.3a through 5.2.3c are for the geodesic. The traversal times for these paths are 1.782, 1.796, and 1.588 seconds respectively, showing that the geodesic does indeed have the shortest traversal time.

When the robot is driven along a given path in minimum time, one or more of the actuators will be driven to its limit. For the straight line path, the  $r$  joint is driven at its maximum or minimum voltage except for two short intervals when the  $\theta$  joint is saturated. For the joint interpolated curve, the  $r$  joint motor voltage is always the limiting factor. For the geodesic, the  $r$  joint is saturated most of the time, but both the  $\theta$  and  $z$  joints are driven to their limits at one time or another. The geodesic seems to distribute the workload more evenly among the joints than the other two curves do.

Two methods (excluding the special case) have been proposed for finding geometric paths which allow a robotic manipulator to move from one point to another in minimum time or approximately minimum time; if obstacle avoidance is not a consideration, both methods yield the same result. While these methods do not directly address the problem of obstacle avoidance, they do demonstrate that the problem of choosing minimum time paths is not simple, and in particular they show that minimum time is not in general equivalent to minimum Cartesian distance.

Two approaches to the obstacle avoidance problem suggest themselves. If the geodesic which connects the desired initial and final positions of the manipulator happens to pass through an obstacle, then we may piece together geodesics to give a path which has shortest geodesic, rather than Cartesian, distance. This again has the disadvantage that the path will have corners at which the manipulator must stop,

but these corners could presumably be rounded off, as in [22].

On the other hand, Eq. (5.1.3.1) provides a means of evaluating the “goodness” of any given path without actually calculating the path’s traversal time. If several paths can be found which avoid collisions with obstacles, then each one can be evaluated and the best one chosen on the basis of formula (5.1.3.1). This presumes that some method can be developed for generating collision-free paths quickly. It also presumes that at least some of the paths generated by the algorithm are reasonably close to the optimal path. But since minimization of the product of curvature and distance gives paths with short traversal times, some guidelines for generating paths are now available.

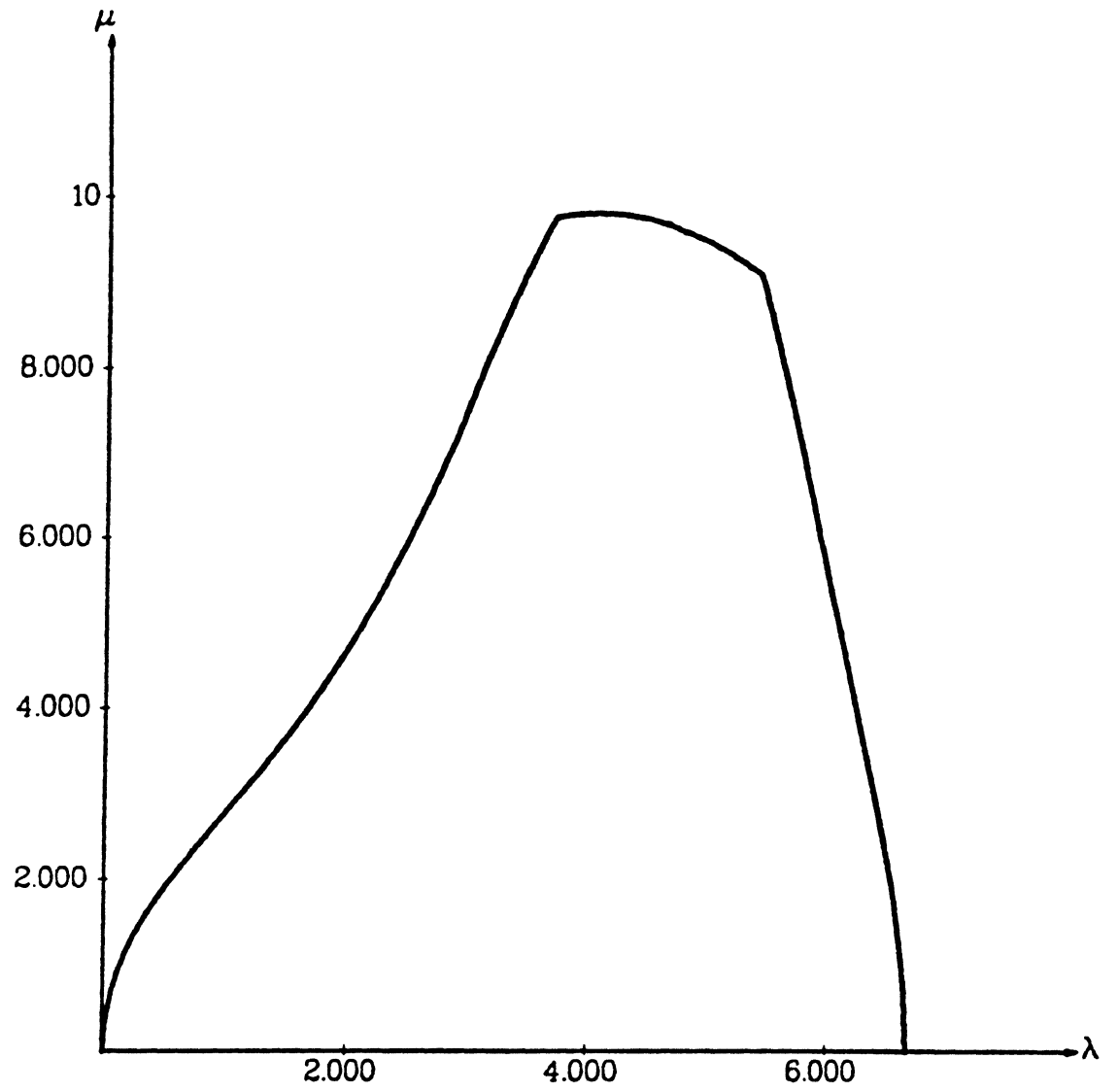


Figure 5.2.1a. Phase plane plot for straight line

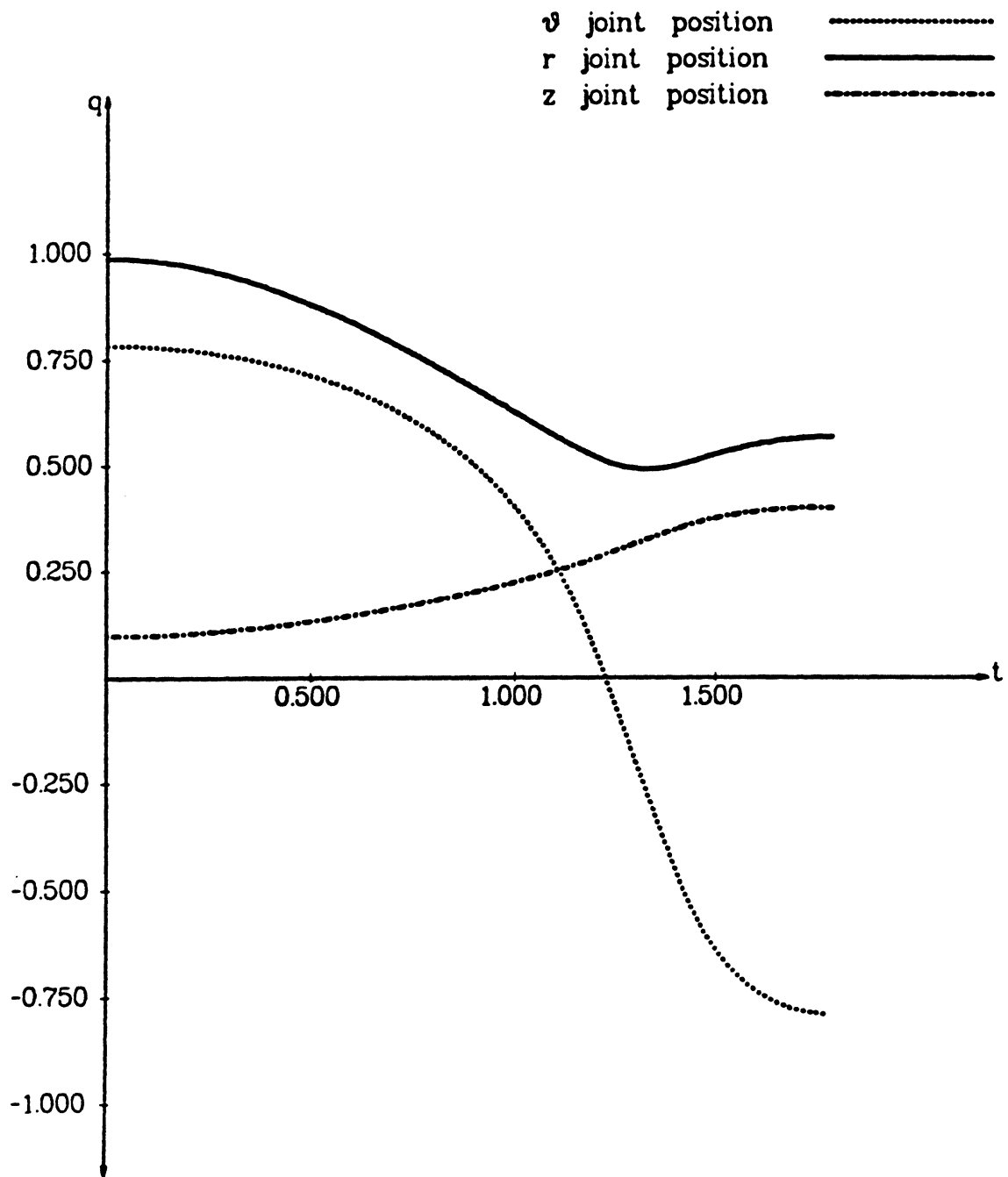


Figure 5.2.1b. Joint position vs. time for straight line

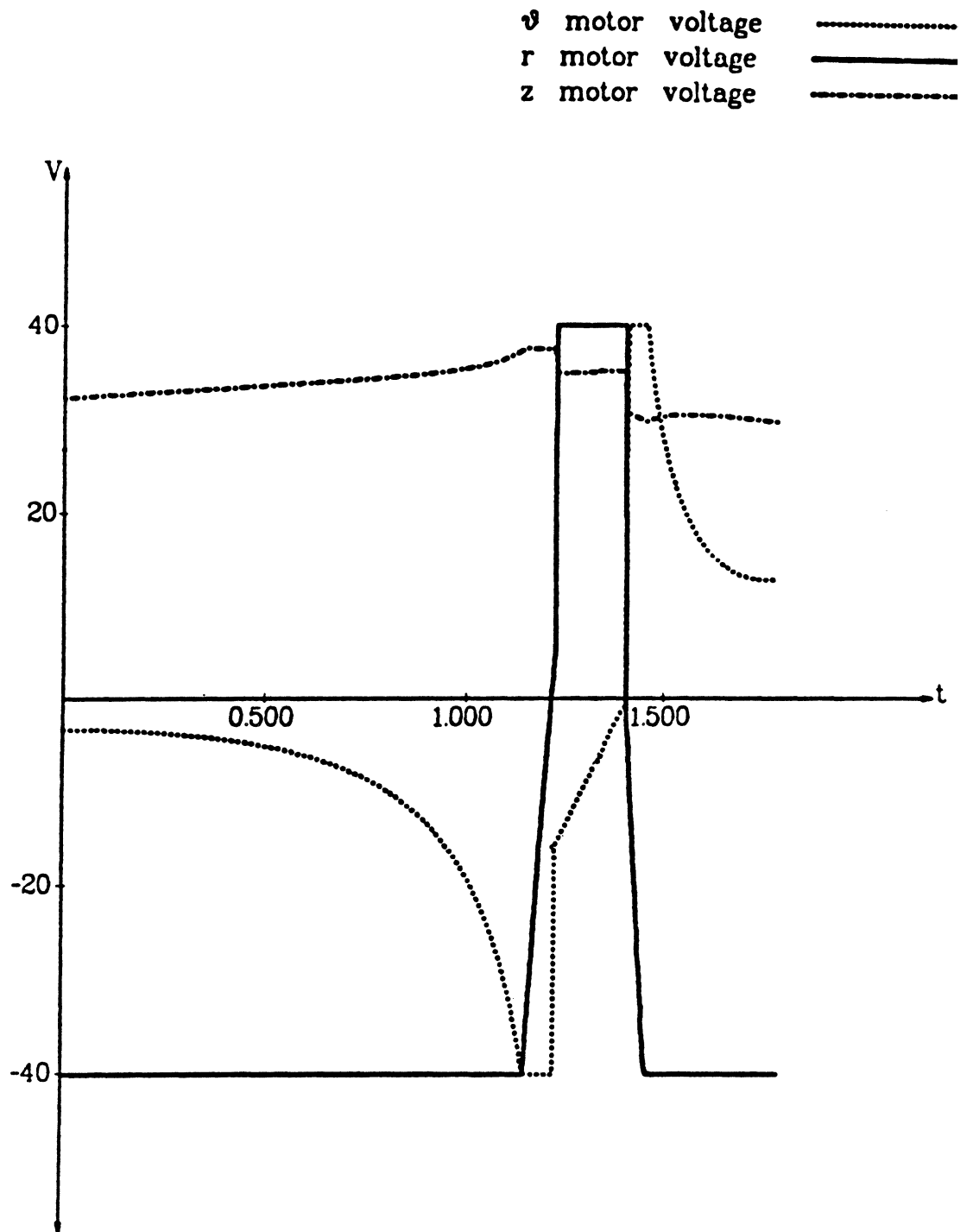


Figure 5.2.1c. Motor voltage vs. time for straight line

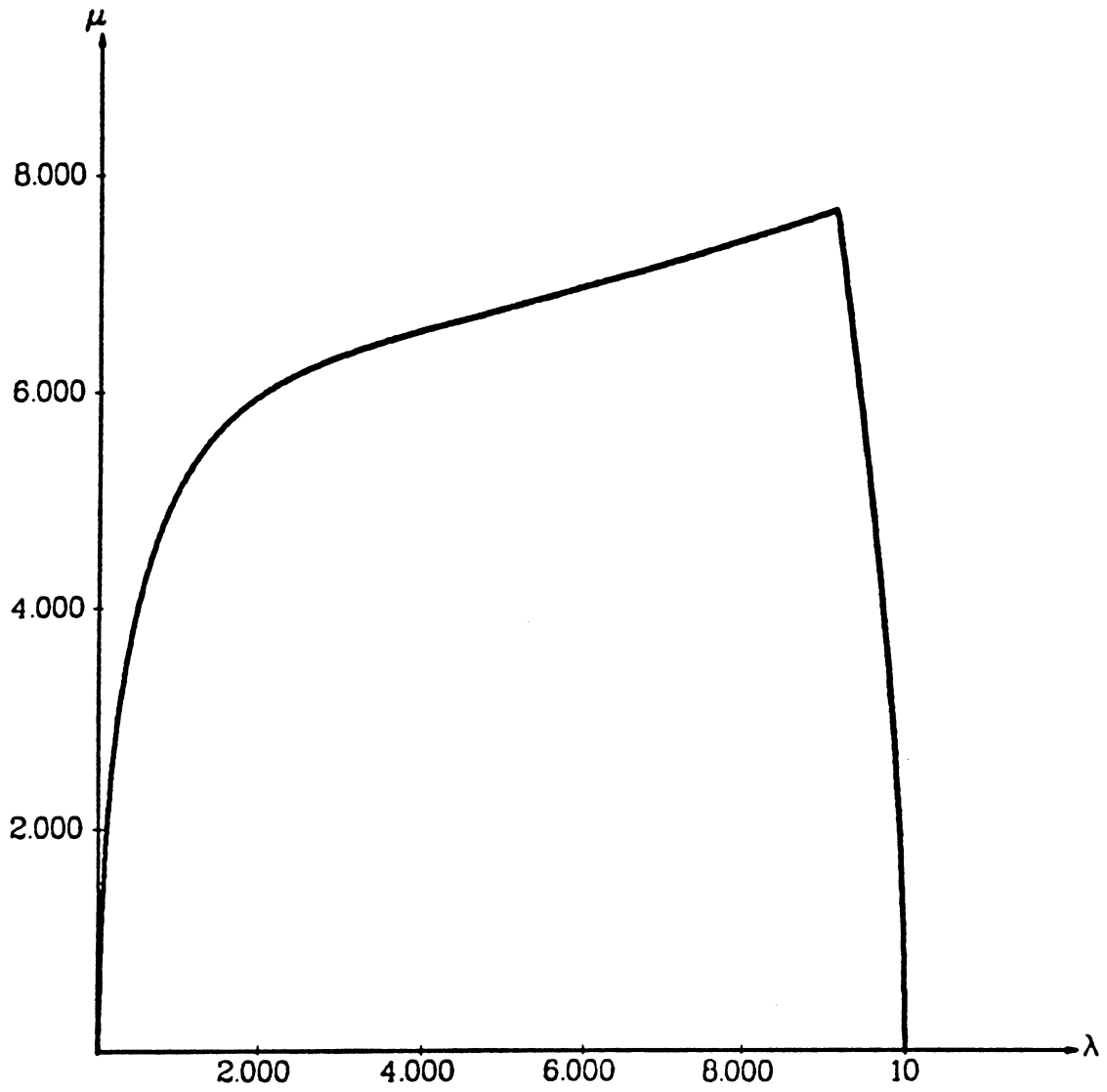


Figure 5.2.2a. Phase plane plot for joint-interpolated path

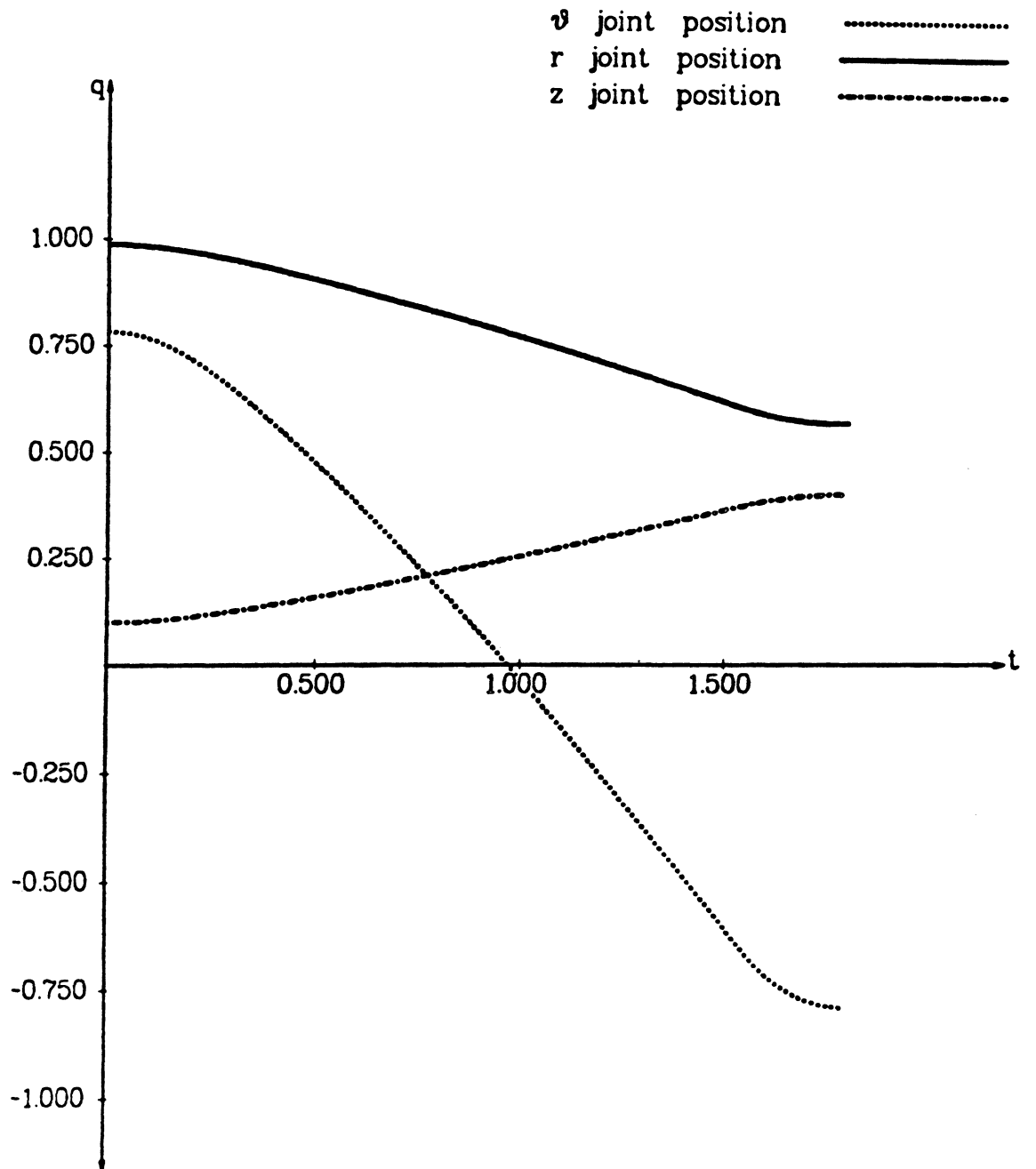


Figure 5.2.2b. Joint position vs. time for joint-interpolated path



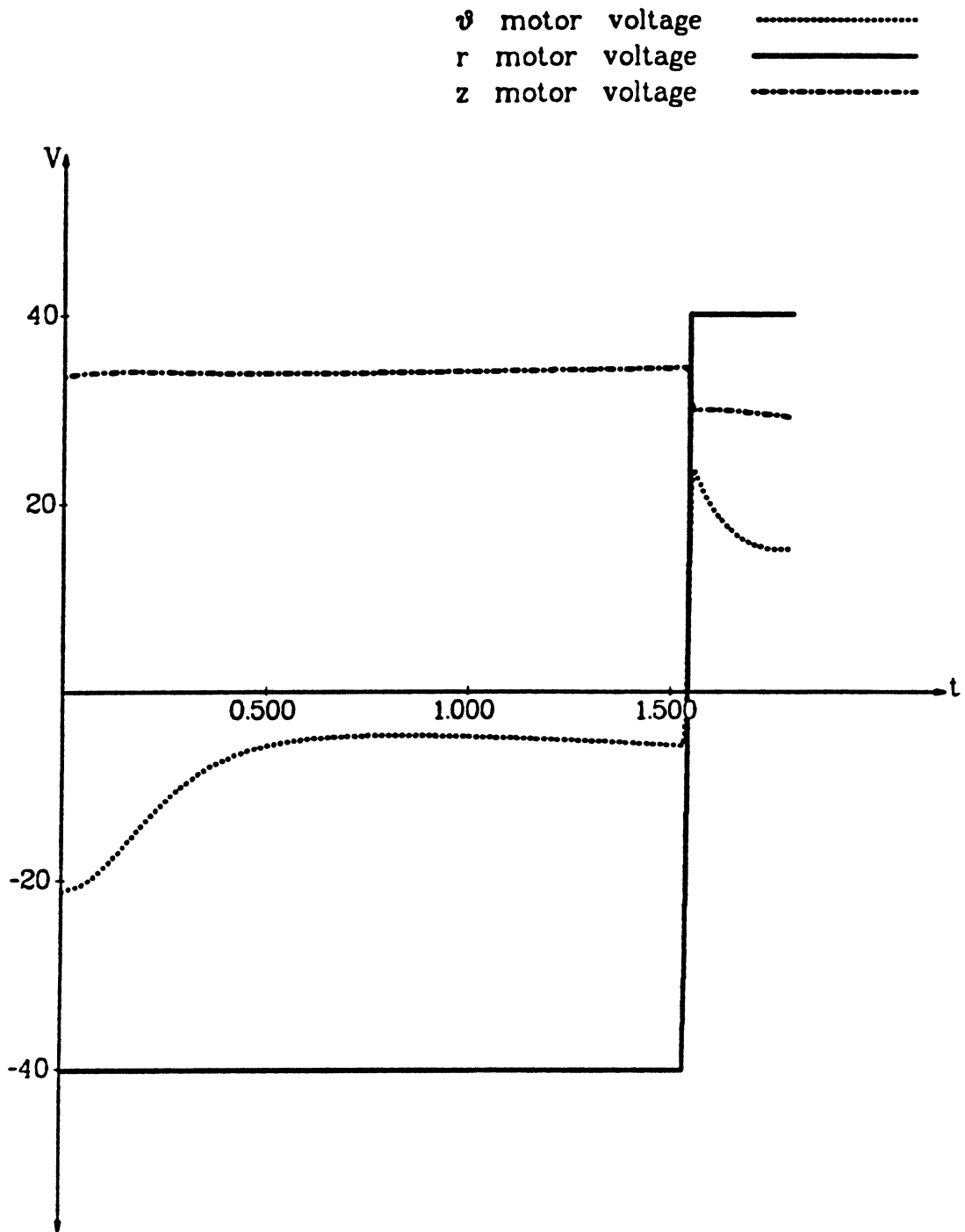


Figure 5.2.2c. Motor voltage vs. time for joint-interpolated path

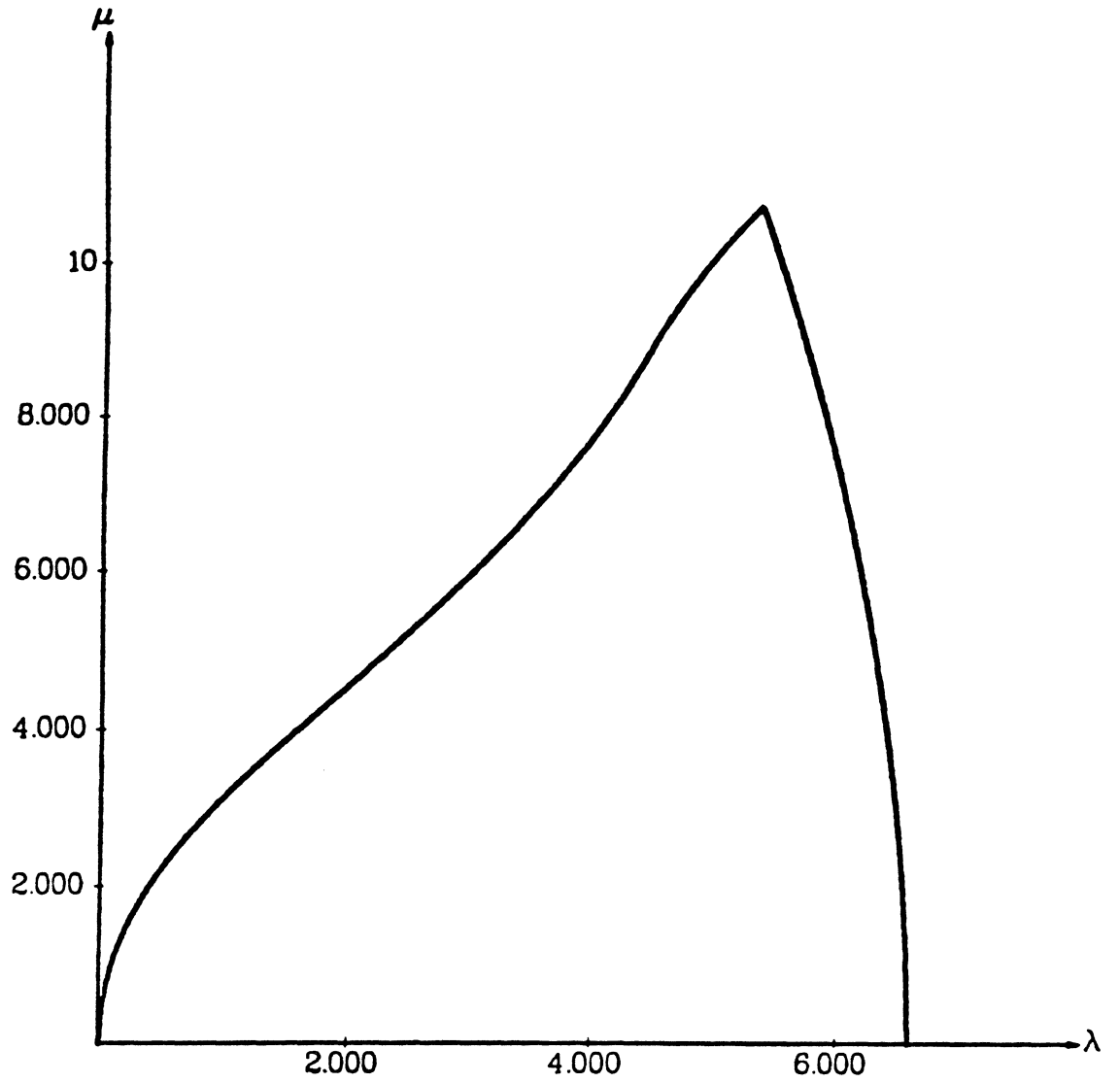


Figure 5.2.3a. Phase plane plot for geodesic

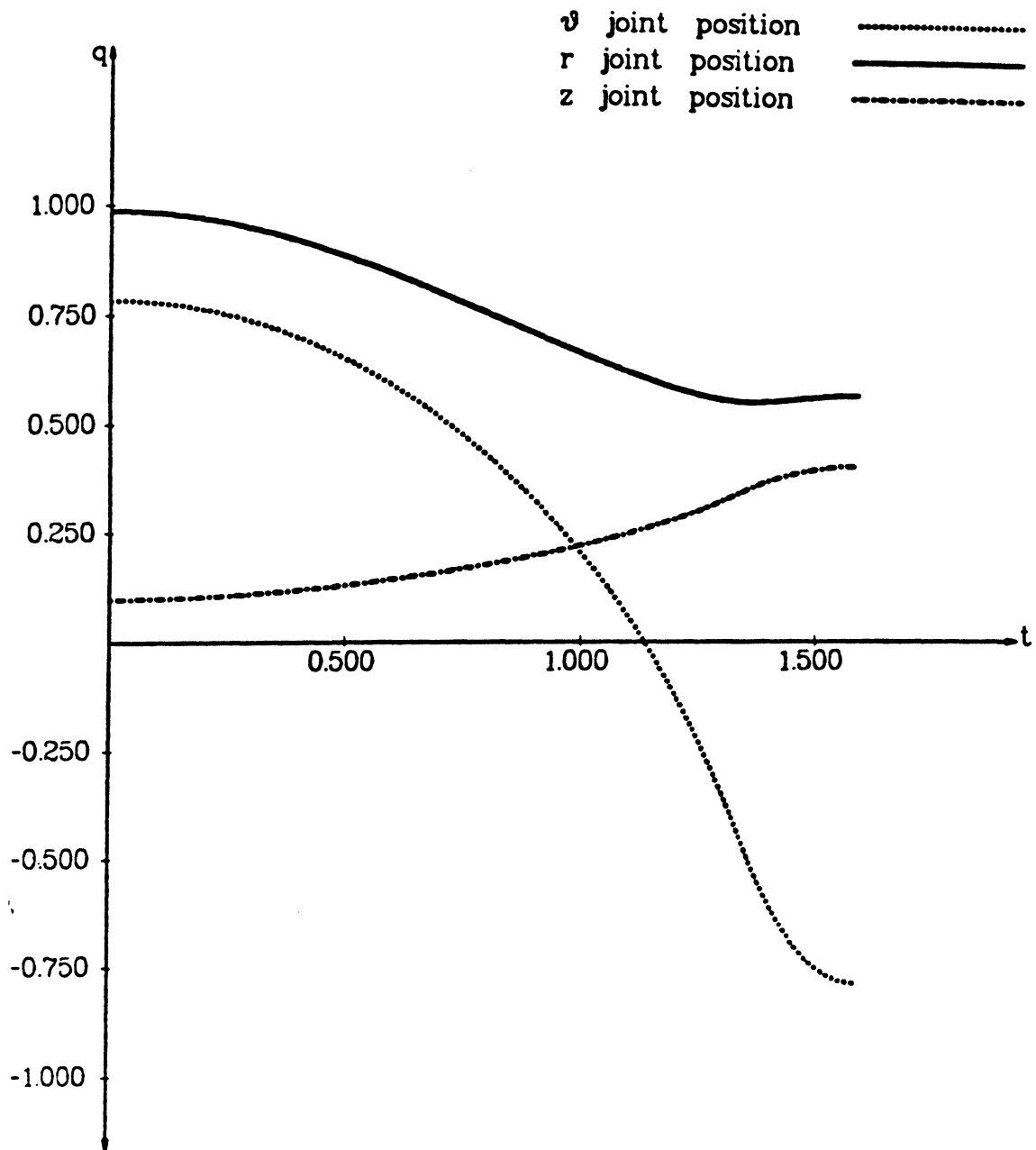


Figure 5.2.3b. Joint position vs. time for geodesic

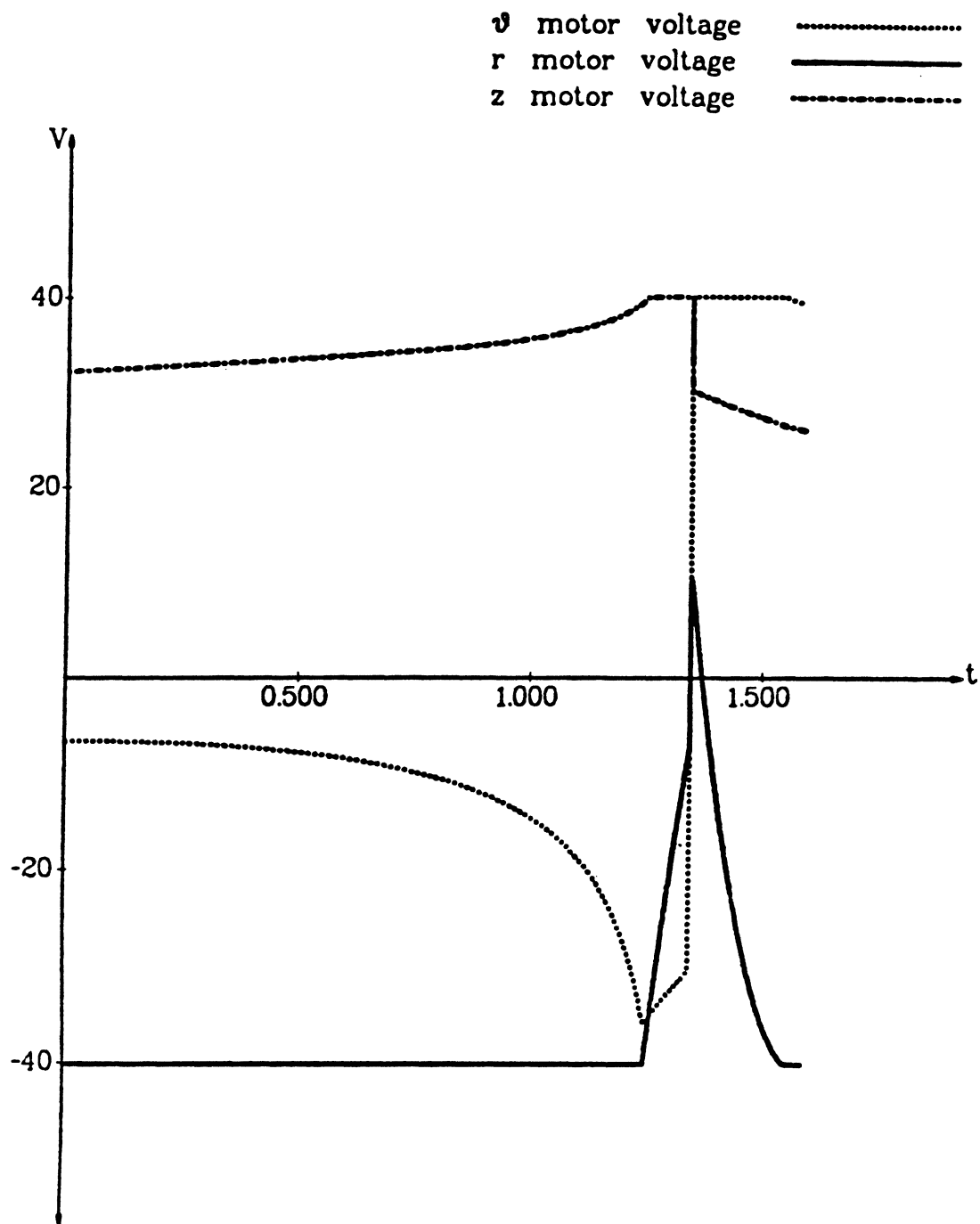


Figure 5.2.3c. Motor voltage vs. time for geodesic

## CHAPTER 6

### COMPENSATION FOR DYNAMIC UNCERTAINTIES

In the previous chapters, it was assumed that the dynamics of the robot were known exactly. In practice, though, this will not be the case. Uncertainties in link inertias and payload characteristics will cause deviations from the nominal dynamic model. If the robot's actuators do not saturate and the dynamic errors are small, then the path tracker can compensate for these dynamic uncertainties. However, the minimum-time trajectory planners described in chapter 4 always generate nominal torques which are at the limits of the robot's capabilities *for the given dynamics*. Moving the robot along the desired path at speeds calculated for the nominal payload may therefore require torques which are beyond the robot's capabilities if the payload differs from the nominal one. If the robot's joints are controlled by independent servoes, as is usually the case, then attempting to make the robot move along the nominal trajectory will result in one or more joints "falling behind", so that the robot strays from the desired geometric path. In other words, the trajectory generated by the trajectory planner is realizable for the nominal dynamics, but not for the actual dynamics.

Though link inertias vary somewhat from one robot to the next, a more important source of dynamic variations is uncertainty in payload mass, shape, and grip

position. The techniques described in this chapter are applicable to variations in link inertias as well as to variations in payload characteristics, but for the sake of simplicity, only compensation for payload characteristics will be described.

There are a number of adaptive controllers which can compensate for the changes in load, provided that the plant (i.e., the robot joint drive) does not saturate [7, 17]. However if the plant saturates, as may happen if the actual and nominal payloads differ too much, then these controllers cannot possibly compensate for load changes. This chapter presents an analysis of the torque errors caused by payload changes, and shows how to incorporate the error information into the trajectory planning process so as to avoid saturation of the individual actuators.

Changes in payload characteristics will be expressed as errors in the *pseudo-inertia* of the payload; the pseudo-inertia is a matrix containing the mass and first and second moments of the payload. It will be shown that bounds on the joint torque errors can be calculated in terms of the norm of the error in the pseudo-inertia of the payload, given the robot's kinematics. Either the dynamic programming trajectory planning algorithm or the perturbation trajectory planner can then be modified to handle uncertainties in the dynamics caused by the payload. If the actual and nominal payloads are described by the pseudo-inertias  $I_A$  and  $I_N$  respectively, then for a given positive real number  $E$ , the algorithm generates a trajectory which is realizable for *all* payloads  $I_A$  for which  $\|I_A - I_N\| \leq E$ .

In order to avoid excessive torque requirements, we wish to compute a set of velocities and accelerations  $\mu$  and  $\dot{\mu}$  such that if the nominal torques  $u_i$  are given by

$$u_i = M_i(I_N)\dot{\mu} + Q_i(I_N)\mu^2 + R_i\mu + S_i(I_N) \quad (6.1)$$

then the actual torques  $\mathbf{u}'_i$  given by

$$\begin{aligned} \mathbf{u}'_i = & M_i(I_N + \Delta I_N)\dot{\mu} + Q_i(I_N + \Delta I_N)\mu^2 \\ & + R_i\mu + S_i(I_N + \Delta I_N) \end{aligned} \quad (6.2)$$

will be realizable, i.e.,

$$\mathbf{u}_i^{\min}(\lambda, \mu) \leq \mathbf{u}'_i \leq \mathbf{u}_i^{\max}(\lambda, \mu). \quad (6.3)$$

Formally, the *Robust Trajectory Planning (RTP)* problem may be stated as follows:

Given a geometric path described as a parameterized curve, the torque limits  $\mathbf{u}_i^{\min}$  and  $\mathbf{u}_i^{\max}$  as functions of  $\lambda$  and  $\mu$ , the dynamics of the robot when carrying the nominal payload  $I_N$ , and a bound  $E$  on the norm of the difference between the pseudo-inertias of the actual and nominal payloads, determine the fastest trajectory (sequence of  $(\lambda, \mu)$  pairs) such that the torques  $\mathbf{u}'_i$  given by (6.2) satisfy the constraints (6.3) for all points on the trajectory and for all payload errors  $\Delta I_N$  such that  $\|\Delta I_N\| \leq E$ .

We will solve this problem by calculating the worst-case torque error, as a function of  $\lambda$ ,  $\mu$ , and  $\dot{\mu}$ , for a given payload error, and decreasing the torque limits by this amount when doing trajectory planning.

### 6.1. Calculation of Dynamic Coefficient Errors

For a given path, we need to know the changes to the coefficients  $M_i$ ,  $Q_i$ ,  $R_i$ , and  $S_i$  in Eq. (6.1) which result from changes in the dynamics of the robot. In the sequel, changes in dynamics will be assumed to come from changes in payload characteristics. While changes in friction coefficients, and hence changes to  $R_i$ , also contribute to changes in required torques, such changes are independent of changes in payload characteristics, and for the sake of simplicity will not be dealt with here.

Determining the change to  $M_i$ , we have

$$M_i(I_N) = \mathbf{J}_{ij}(I_N) \frac{df^j}{d\lambda} \quad (6.1.1)$$

and

$$M_i(I_N + \Delta I_N) = \mathbf{J}_{ij}(I_N + \Delta I_N) \frac{df^j}{d\lambda} \quad (6.1.2)$$

so that

$$\begin{aligned} \delta M_i(I_N, \Delta I_N) &\equiv M_i(I_N + \Delta I_N) - M_i(I_N) \\ &= \left\{ \mathbf{J}_{ij}(I_N + \Delta I_N) - \mathbf{J}_{ij}(I_N) \right\} \frac{df^j}{d\lambda} \equiv \delta \mathbf{J}_{ij}(I_N, \Delta I_N) \frac{df^j}{d\lambda}. \end{aligned} \quad (6.1.3)$$

Differences between nominal and actual payload characteristics cause changes in the coefficients  $\mathbf{J}_{ij}$ ,  $\mathbf{C}_{ijk} \equiv [jk, i]$ , and  $\mathbf{g}_i$  in equation (2.4.12). Here we determine the relationship between changes in these coefficients and changes in payload characteristics.

Changes in payload characteristics will result in changes to the pseudo-inertia tensor of the last joint of the robot, i.e., the pseudo-inertia tensor will have the value  $I_N + \Delta I_N$  instead of  $I_N$ . In order to obtain  $\delta M_i$ , we consider how this affects the inertia matrix. The coefficients in the inertia matrix are given in [28] as

$$\mathbf{J}_{ij}(I_N) = \sum_{p=\max(i,j)}^N \text{Tr} \left( \frac{\partial \mathbf{T}_p}{\partial \mathbf{q}^j} I_p \frac{\partial \mathbf{T}_p^T}{\partial \mathbf{q}^i} \right) \quad (6.1.4)$$

where  $\mathbf{T}_p$  is the  $4 \times 4$  homogeneous transformation matrix which transforms vectors given in the coordinate system associated with the  $p^{\text{th}}$  link of the robot to world or



base coordinates, and  $I_p$  is the pseudo-inertia of the  $p^{\text{th}}$  link given in the  $p^{\text{th}}$  link's coordinate frame. The pseudo-inertia is defined [28] as the matrix

$$I_p = \begin{bmatrix} \int x^2 dm & \int xy dm & \int xz dm & \int x dm \\ \int xy dm & \int y^2 dm & \int yz dm & \int y dm \\ \int xz dm & \int yz dm & \int z^2 dm & \int z dm \\ \int x dm & \int y dm & \int z dm & \int dm \end{bmatrix}. \quad (6.1.5)$$

The coordinates  $x$ ,  $y$ , and  $z$  are expressed in the  $p^{\text{th}}$  coordinate frame, and the integrals are all taken over the  $p^{\text{th}}$  link.

Introducing an error  $\Delta I_N$  into the pseudo-inertia of the last joint gives

$$\begin{aligned} \mathbf{J}_{ij}(I_N + \Delta I_N) = & \sum_{p=\max(i,j)}^{N-1} \text{Tr} \left( \frac{\partial \mathbf{T}_p}{\partial \mathbf{q}^j} I_p \frac{\partial \mathbf{T}_p^T}{\partial \mathbf{q}^i} \right) \\ & + \text{Tr} \left( \frac{\partial \mathbf{T}_N}{\partial \mathbf{q}^j} (I_N + \Delta I_N) \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right). \end{aligned} \quad (6.1.6)$$

Subtracting (6.1.4) from (6.1.6) gives

$$\delta \mathbf{J}_{ij}(I_N, \Delta I_N) = \delta \mathbf{J}_{ij}(\Delta I_N) = \text{Tr} \left( \frac{\partial \mathbf{T}_N}{\partial \mathbf{q}^j} \Delta I_N \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right). \quad (6.1.7)$$

Note that the error in the inertia matrix is linear in the pseudo-inertia error, and is *independent* of the nominal payload. To find  $\delta M_i$ , simply plug (6.1.7) into (6.1.3), giving

$$\delta M_i = \sum_j \frac{df^j}{d\lambda} \text{Tr} \left( \frac{\partial \mathbf{T}_N}{\partial \mathbf{q}^j} \Delta I_N \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right) \quad (6.1.8)$$

Computation of the errors  $\delta Q_i$  follows the same pattern as the computation of  $\delta M_i$ . The errors in the Coriolis terms can be determined in much the same way as the errors in the inertia matrix. From [28] we have

$$\mathbf{C}_{ijk} = \sum_{p=\max(i,j,k)}^N \text{Tr} \left( \frac{\partial^2 \mathbf{T}_p}{\partial \mathbf{q}^j \partial \mathbf{q}^k} I_p \frac{\partial \mathbf{T}_p^T}{\partial \mathbf{q}^i} \right). \quad (6.1.9)$$

The errors in the Coriolis terms due to errors in payload characteristics are therefore given by

$$\delta \mathbf{C}_{ijk} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_N}{\partial \mathbf{q}^j \partial \mathbf{q}^k} \Delta I_N \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right). \quad (6.1.10)$$

The definition of  $Q_i$  gives

$$\begin{aligned} \delta Q_i = \sum_j \text{Tr} \left( \frac{\partial \mathbf{T}_N}{\partial \mathbf{q}^j} \Delta I_N \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right) \frac{d^2 f^j}{d\lambda^2} \\ + \sum_j \sum_k \text{Tr} \left( \frac{\partial^2 \mathbf{T}_N}{\partial \mathbf{q}^j \partial \mathbf{q}^k} \Delta I_N \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right) \frac{df^j}{d\lambda} \frac{df^k}{d\lambda}. \end{aligned} \quad (6.1.11)$$

Now we need to know the error in the gravitational terms. The gravitational forces  $\mathbf{g}_i$  are given by [28]

$$\mathbf{g}_i = \sum_{k=i}^N -m_k \mathbf{G}^T \frac{\partial \mathbf{T}_k}{\partial \mathbf{q}^i} \bar{\mathbf{r}}_k \quad (6.1.12)$$

where

$$\mathbf{G} = \begin{bmatrix} 0 \\ 0 \\ g \\ 0 \end{bmatrix}$$

is the gravitational force vector,  $m_k$  is the mass of the  $k^{\text{th}}$  link,  $g$  is the acceleration due to gravity, and

$$\bar{\mathbf{r}}_k = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \\ 1 \end{bmatrix}$$

is the center of mass of the  $k^{\text{th}}$  link given in the coordinates of the  $k^{\text{th}}$  frame. If we define  $\mathbf{w}_k \equiv m_k \bar{\mathbf{r}}_k$ , then we have

$$\mathbf{g}_i = -\sum_{k=i}^N \mathbf{G}^T \frac{\partial \mathbf{T}_k}{\partial \mathbf{q}^i} \mathbf{w}_k. \quad (6.1.13)$$

But  $\mathbf{w}_k$  is just the last column of the pseudo-inertia matrix  $I_k$ , so that

$$\mathbf{g}_i = -\sum_{k=i}^N \mathbf{G}^T \frac{\partial \mathbf{T}_k}{\partial \mathbf{q}^i} I_k \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (6.1.14)$$

As before, introducing an error into the pseudo-inertia of the last link gives

$$\delta S_i = \delta \mathbf{g}_i = -\mathbf{G}^T \frac{\partial \mathbf{T}_N}{\partial \mathbf{q}^i} \Delta I_N \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (6.1.15)$$

We may now calculate the error in  $\mathbf{u}_i$  by adding up the individual components, giving

$$\delta u_i = \delta M_i \mu + \delta Q_i \mu^2 + \delta S_i. \quad (6.1.16)$$

## 6.2. Calculation of Torque Error Bounds

If the errors  $\Delta I_N$  were known exactly, then the torque errors could also be computed exactly. Of course, in practice  $\Delta I_N$  will not be known exactly. However, if bounds on the norm of  $\Delta I_N$  can be obtained, then we may find bounds on  $\delta u_i$ .

To obtain these bounds, note that  $\delta M_i$ ,  $\delta Q_i$ , and  $\delta S_i$  are all functions of the pseudo-inertia error  $\Delta I_N$ ; in fact, they are *linear* in  $\Delta I_N$ , so that  $\delta u_i$  is also linear in  $\Delta I_N$ . If we write

$$\delta u_i = Z(\Delta I_N), \quad (6.2.1)$$

then we wish to maximize or minimize the linear function  $Z$  with respect to  $\Delta I_N$ , subject to

$$\|\Delta I_N\| \leq E \quad (6.2.2)$$

where  $E$  is the bound on the pseudo-inertia error.

At this point, some observations are in order. First, as we noted before, the errors in the  $\delta u_i$  depend linearly upon the pseudo-inertia error  $\Delta I_N$ . Second, note also that  $\delta u_i$  depends *only* on the kinematics of the robot and on the desired velocity and acceleration, not on the nominal dynamics. These facts are consequences of the fact that both kinetic and potential energy are linear in mass. To see this, consider the form of the Lagrangian, namely  $L = K - P$ , where  $K$  and  $P$  are the kinetic and potential energies of the robot. Any difference between the actual and nominal

dynamics of the robot will cause a corresponding difference in the Lagrangian. If we think of the error in the pseudo-inertia as a piece of stray material stuck to the end effector, then both the kinetic and potential energies of this material, and therefore the Lagrangian, depend only upon the mass and shape of the stray material and upon the position and speed of the end effector. The Lagrangian is linear in the mass of the material; since Lagrange's differential equations are linear in  $L$ , the errors in the generalized forces can be separated from the forces resulting from the nominal mass of the manipulator. One consequence of this separability of forces is that the errors  $\delta u_i$  do not depend upon the nominal dynamics, as shown above. The implication of this is that much of the error analysis can proceed without regard to the nominal dynamics of the robot.

The linearity of the  $\delta u_i$  in the pseudo-inertia has some other practical consequences as well. Consider the maximization which must be performed in order to evaluate  $\delta u_i$ . This maximization requires that the space of  $4 \times 4$  symmetric matrices with norm less than  $E$  be searched, which in general is a rather formidable problem. However, by choosing a particular class of matrix norms, the problem can be made quite simple; in fact, it can be transformed into a linear programming problem.

To see how this transformation can be performed, consider the problem of maximizing the function  $Z$  in equation (6.2.1), namely

$$\text{Problem A: maximize } Z(\mathbf{M}) = \sum_i \sum_j \beta_{ij} \mathbf{M}_{ij} \quad (6.2.3)$$

$$\text{subject to } \|\mathbf{M}\| \leq E \text{ and } \mathbf{M} = \mathbf{M}^T. \quad (6.2.4)$$

Treatment of the minimization problem proceeds analogously to problem A. We will show that problem A transforms into a linear programming problem if the norm used to constrain the matrix  $\mathbf{M}$  in (6.2.4) is chosen properly. This will be accomplished by eliminating some absolute values from the constraints.

$Z(\mathbf{M})$ , the function to be maximized, is a linear function of  $\mathbf{M}$ . It remains to be shown that the constraints can be made linear. Of course if the norm used in problem A is arbitrary, then in general the constraints will not be linear. However, there is a set of norms, all very easy to calculate, which will yield linear constraints. Consider the class of functions  $F: \mathbf{R}^{4 \times 4} \rightarrow \mathbf{R}^+$  given by

$$F(\mathbf{M}) = \max_i \sigma_i(\mathbf{M})$$

where

$$\sigma_i(\mathbf{M}) = \sum_{j=1}^4 \sum_{k=1}^4 \alpha_{ijk} |M_{jk}|.$$

The matrix 1-norm and  $\infty$ -norm,  $\max_{i,j} |M_{ij}|$  and  $\sum_{i,j} |M_{ij}|$  are all functions of this form. We now show that, under suitable conditions,  $F$  is a norm.

**Lemma 6.1:** If  $\alpha_{ijk} \geq 0$  for all  $i, j$  and  $k$ , and if for every pair of indices  $(j, k)$  there is an  $i$  such that  $\alpha_{ijk} \neq 0$ , then  $F(\mathbf{M})$  is a norm.

**Proof:** In order to prove that  $F$  is a norm, we must show that

- 1)  $F(\mathbf{M}) \geq 0$  for all  $\mathbf{M}$ ,
- 2)  $F(\mathbf{M}) = 0$  iff  $\mathbf{M} = 0$ ,
- 3)  $F(\gamma\mathbf{M}) = |\gamma| F(\mathbf{M})$  for all scalars  $\gamma$  and all matrices  $\mathbf{M}$ , and
- 4)  $F(\mathbf{X} + \mathbf{Y}) \leq F(\mathbf{X}) + F(\mathbf{Y})$  for all matrices  $\mathbf{X}$  and  $\mathbf{Y}$ .

Obviously (1) is true, since  $F$  is the maximum of a set of non-negative quantities.  $F(0) = 0$ , proving the "if" part of (2). To prove the "only if" part, observe that if  $\mathbf{M}$  is non-zero, then it has some non-zero element  $M_{jk}$ . For this particular  $jk$  pair, there is some  $i$  such that  $\alpha_{ijk}$  is non-zero, so that  $\sigma_i > 0$  for this  $i$ . Therefore  $F > 0$ . (3) is true, since

$$\sigma_i(\gamma\mathbf{M}) = \sum_{j=1}^4 \sum_{k=1}^4 \alpha_{ijk} |\gamma M_{jk}| = |\gamma| \sum_{j=1}^4 \sum_{k=1}^4 \alpha_{ijk} |M_{jk}| = |\gamma| \sigma_i(\mathbf{M})$$

and hence

$$F(\gamma\mathbf{M}) = \max_i \sigma_i(\gamma\mathbf{M}) = \max_i |\gamma| \sigma_i(\mathbf{M}) = |\gamma| \max_i \sigma_i(\mathbf{M}) = |\gamma| F(\mathbf{M}).$$

Finally,

$$\begin{aligned} \sigma_i(\mathbf{X} + \mathbf{Y}) &= \sum_j \sum_k \alpha_{ijk} |\mathbf{X}_{jk} + \mathbf{Y}_{jk}| \\ &\leq \sum_j \sum_k \alpha_{ijk} |\mathbf{X}_{jk}| + \sum_j \sum_k \alpha_{ijk} |\mathbf{Y}_{jk}| = \sigma_i(\mathbf{X}) + \sigma_i(\mathbf{Y}) \end{aligned}$$

so that

$$\begin{aligned} F(\mathbf{X} + \mathbf{Y}) &= \max_i \left\{ \sigma_i(\mathbf{X} + \mathbf{Y}) \right\} \leq \max_i \left\{ \sigma_i(\mathbf{X}) + \sigma_i(\mathbf{Y}) \right\} \\ &\leq \max_i \left\{ \sigma_i(\mathbf{X}) \right\} + \max_i \left\{ \sigma_i(\mathbf{Y}) \right\} = F(\mathbf{X}) + F(\mathbf{Y}) \end{aligned}$$

■

Problem A with this class of norms becomes

$$\text{Problem B: maximize } Z(\mathbf{M}) = \sum_i \sum_j \beta_{ij} \mathbf{M}_{ij} \quad (6.2.5)$$

$$\text{subject to } \max_i \left( \sum_{j=1}^4 \sum_{k=1}^4 \alpha_{ijk} |\mathbf{M}_{jk}| \right) \leq E \text{ and } \mathbf{M}_{jk} = \mathbf{M}_{kj}$$

This problem obviously is equivalent to

$$\text{Problem C: maximize } Z(\mathbf{M}) = \sum_i \sum_j \beta_{ij} \mathbf{M}_{ij} \quad (6.2.6)$$

$$\text{subject to } \sum_{j=1}^4 \sum_{k=1}^4 \alpha_{ijk} |\mathbf{M}_{jk}| \leq E \quad \forall i \text{ and } \mathbf{M}_{jk} = \mathbf{M}_{kj}.$$

Problem C may be transformed into a standard linear programming problem by making the substitutions  $\mathbf{M}_{ij} = P_{ij} - N_{ij}$  and  $|\mathbf{M}_{ij}| = P_{ij} + N_{ij}$ , where  $P_{ij}$  and  $N_{ij}$  are non-negative real numbers. To prove that this substitution gives the correct result, first eliminate the symmetry constraint, giving

$$\text{Problem C': maximize } W(\mathbf{M}) = \sum_{j=1}^4 \sum_{k=j}^4 \beta'_{jk} \mathbf{M}_{jk}$$

$$\text{subject to } \sum_{j=1}^4 \sum_{k=j}^4 \alpha'_{ijk} |\mathbf{M}_{jk}| \leq E$$

where

$$\alpha'_{ijk} = \begin{cases} \alpha_{ijk} & j=k \\ \alpha_{ijk} + \alpha_{ikj} & j \neq k \end{cases}$$



and

$$\beta_{ij} = \begin{cases} \beta_{ij} & i=j \\ \beta_{ij} + \beta_{ji} & i \neq j \end{cases}$$

Then we have the following theorem:

**Theorem 6.1:** Let problem D be defined as:

$$\text{maximize } Z(P, N) = \sum_{j=1}^4 \sum_{k=j}^4 \beta'_{jk} (P_{jk} - N_{jk})$$

$$\text{subject to } \sum_{j=1}^4 \sum_{k=j}^4 \alpha'_{ijk} (P_{jk} + N_{jk}) \leq E \text{ and } P_{ij} \geq 0, N_{ij} \geq 0.$$

Then the optimal  $W$  from problem  $C'$  is equal to the optimal  $Z$  from problem D.

**Proof:** Let  $M^*$  be a solution of problem  $C'$ , and let  $W^* = W(M^*)$ . If we make the substitutions

$$P_{jk} = \begin{cases} M_{jk}^* & M_{jk}^* \geq 0 \\ 0 & M_{jk}^* < 0 \end{cases}$$

and

$$N_{jk} = \begin{cases} 0 & M_{jk}^* \geq 0 \\ -M_{jk}^* & M_{jk}^* < 0 \end{cases}$$

then we have  $M_{ij}^* = P_{ij} - N_{ij}$ , and  $|M_{ij}^*| = P_{ij} + N_{ij}$ . Making these substitutions in problem  $C'$  gives

$$W^* = W(M^*) = \sum_{j=1}^4 \sum_{k=j}^4 \beta'_{jk} (P_{jk} - N_{jk}) = Z(P, N)$$

$$\sum_{j=1}^4 \sum_{k=j}^4 \alpha'_{ijk} (P_{jk} + N_{jk}) \leq E \text{ and } P_{ij} \geq 0, N_{ij} \geq 0.$$

The conditions for problem D are satisfied, so we must have  $W^* \leq Z^*$ , where  $Z^*$  is the optimal value of  $Z$  obtained from problem D.

Likewise, let  $P^*$  and  $N^*$  be an optimal solution to problem D. Then for every pair of indices  $(j, k)$  we have  $\beta'_{jk} > 0$ ,  $\beta'_{jk} = 0$ , or  $\beta'_{jk} < 0$ . If  $\beta'_{jk} > 0$ , then we must have  $N_{jk}^* = 0$ . Otherwise, we could substitute  $P_{jk}^* + N_{jk}^*$  for  $P_{jk}$  and 0 for  $N_{jk}$ ; these new values still satisfy the required constraints, but increase the objective function, contradicting the fact that  $(P^*, N^*)$  is optimal. Similarly, if  $\beta'_{jk} < 0$ , then we must have  $P_{jk}^* = 0$ . If  $\beta'_{jk} = 0$ , then we may take  $P_{jk}^* = N_{jk}^* = 0$ , since this leaves the constraints satisfied and has no effect on the objective function. Therefore we always have either  $P_{jk}^* = 0$  or  $N_{jk}^* = 0$ . Taking  $M_{ij} = P_{ij}^* - N_{ij}^*$ , it follows that  $|M_{ij}| = P_{ij}^* + N_{ij}^*$ . Making these substitutions in problem D,

$$Z^* = Z(P^*, N^*) = \sum_{j=1}^4 \sum_{k=j}^4 \beta'_{jk} M_{jk} \leq W^*$$

$$\sum_{j=1}^4 \sum_{k=j}^4 \alpha'_{ijk} |M_{jk}| \leq E$$

Therefore  $W^* \leq Z^* \leq W^*$ , proving the theorem. ■

Now that torque error bounds can be obtained from pseudo-inertia errors, these results must be incorporated into the trajectory planning process. Direct use of the results derived above in the phase plane trajectory planner is not easy, since this trajectory planner requires that we solve (6.1) for  $\dot{\mu}$  in terms of  $\lambda$ ,  $\mu$ , and  $u_i$ . This solution is required because the trajectory planner must convert torque ranges into  $\dot{\mu}$  ranges. When errors are introduced, we have

$$\begin{aligned} u_i^{\min}(\lambda, \mu) &\leq M_i \dot{\mu} + Q_i \mu^2 + R_i \mu + S_i + \min_{\|\Delta I_N\| \leq E} \left\{ \delta M_i \dot{\mu} + \delta Q_i \mu^2 + \delta S_i \right\} \\ &\leq M_i \dot{\mu} + Q_i \mu^2 + R_i \mu + S_i + \max_{\|\Delta I_N\| \leq E} \left\{ \delta M_i \dot{\mu} + \delta Q_i \mu^2 + \delta S_i \right\} \\ &\leq u_i^{\max}(\lambda, \mu). \end{aligned}$$

For given  $\lambda$  and  $\mu$ , these inequalities determine a range of values of  $\dot{\mu}$ . However, the worst-case values of  $\delta M_i$ ,  $\delta Q_i$ , and  $\delta S_i$  depend upon  $\dot{\mu}$ , which in turn depends upon the values of  $\delta M_i$ ,  $\delta Q_i$ , and  $\delta S_i$ , so finding the allowable range of values of  $\dot{\mu}$  explicitly is difficult. Of course these equations can be solved numerically. For example, to find the maximum allowable value of  $\dot{\mu}$ , the equation

$$M_i \dot{\mu} + Q_i \mu^2 + R_i \mu + S_i + \max_{\|\Delta I_N\| \leq E} \left\{ \delta M_i \dot{\mu} + \delta Q_i \mu^2 + \delta S_i \right\} = u_i^{\max}(\lambda, \mu)$$

can be solved by bisection for  $\dot{\mu}$ .

A simpler solution to the problem is to do trajectory planning by either the dynamic programming method or the perturbation method. These trajectory

planners only need to have available a test function which determines whether or not a given  $(\lambda, \mu, \dot{\mu})$  triple requires excessive torque; in effect, they automatically perform the numerical search for the allowable values of  $\dot{\mu}$ . But such a function is easily constructed, since for a given  $(\lambda, \mu, \dot{\mu})$  we can easily minimize or maximize  $\delta u_i$  in (6.1.16), and see if  $u_i + \delta u_i^{\max}$  exceeds  $u_i^{\max}(\lambda, \mu)$  or  $u_i + \delta u_i^{\min}$  falls below  $u_i^{\min}(\lambda, \mu)$ . In particular, the following algorithm checks to see if a particular  $(\lambda, \mu, \dot{\mu})$  triple meets all the torque constraints:

**for each joint  $i$  do**

**begin**

$$\text{compute } u_i^N = M_i(\lambda)\dot{\mu} + Q_i(\lambda)\mu^2 + R_i(\lambda)\mu + S_i(\lambda)$$

$$\text{compute } \delta u_i^{\max} = \max_{\|\Delta I_N\| \leq E} \left\{ \delta M_i(\lambda, \Delta I_N)\dot{\mu} + \delta Q_i(\lambda, \Delta I_N)\mu^2 + \delta S_i(\lambda, \Delta I_N) \right\}$$

$$\text{compute } \delta u_i^{\min} = \min_{\|\Delta I_N\| \leq E} \left\{ \delta M_i(\lambda, \Delta I_N)\dot{\mu} + \delta Q_i(\lambda, \Delta I_N)\mu^2 + \delta S_i(\lambda, \Delta I_N) \right\}$$

**if  $u_i^N + \delta u_i^{\min} < u_i^{\min}(\lambda, \mu)$ , then return REJECT**

**if  $u_i^N + \delta u_i^{\max} > u_i^{\max}(\lambda, \mu)$ , then return REJECT**

**end**

**return ACCEPT.**

It should be noted that this function is called for *each*  $(\lambda, \mu)$  pair; it does not, for example, reject a  $(\lambda, \mu, \dot{\mu})$  triple based on an error which is computed for *all* positions or *all* velocities. As a consequence, speed is sacrificed only when absolutely necessary to guarantee that the trajectory will be realizable for all payloads within the allowable range.

### 6.3. Numerical Examples

As an example, we apply the methods of the previous section to the first three joints of the Bendix PACS robot arm. The trajectory planner uses the perturbation algorithm.

The kinematics of the PACS arm are quite simple. If the coordinate frames of the base and hand are as shown in the Appendix in Figure A.2, then the coordinate transform  $\mathbf{T}_3$  is

$$\mathbf{T}_3 = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & -r \sin \theta \\ \sin \theta & 0 & \cos \theta & r \cos \theta \\ 0 & -1 & 0 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.3.1)$$

The calculation of the partial derivatives of  $\mathbf{T}_3$  and the computation of the coefficients  $\delta M_i$ ,  $\delta Q_i$ , and  $\delta S_i$  can be found in the Appendix.

We will use the norm

$$\|\mathbf{H}\| = \sum_{i=1}^4 \sum_{j=1}^4 \alpha_{ij} |\mathbf{H}_{ij}|$$

where  $\alpha_{ij} > 0$ . This makes the problem of finding the error bounds very simple. It is easily seen that if the functional to be maximized is

$$Z = \sum_i \sum_j \beta_{ij} \mathbf{H}_{ij}$$

then the maximum over  $\mathbf{H}$  for  $\|\mathbf{H}\| \leq E$  occurs when all the  $\mathbf{H}_{ij}$  are zero except for those  $\mathbf{H}_{ij}$  for which  $\left| \frac{\beta_{ij}}{\alpha_{ij}} \right|$  is a maximum; this number times  $E$  is also the max-

imum value of  $Z$ . If we use

$$\alpha_{ij} = \begin{cases} 1 & i=j \\ 1/2 & i \neq j \end{cases}$$

then the resulting bounds on the  $|\delta u_i|$  are

$$|\delta u_z| \leq \left| \frac{dz}{d\lambda} \dot{\mu} + \frac{d^2z}{d\lambda^2} \mu^2 + g \right| \|\Delta I_N\|$$

$$|\delta u_\theta| \leq \max \left\{ \left| \frac{d\theta}{d\lambda} \dot{\mu} + \frac{d^2\theta}{d\lambda^2} \mu^2 \right|, \left| \frac{dr}{d\lambda} \dot{\mu} + \frac{d^2r}{d\lambda^2} \mu^2 \right|, \right.$$

$$\left. \left| 2r \frac{d\theta}{d\lambda} \dot{\mu} + 2r \frac{d^2\theta}{d\lambda^2} \mu^2 + 2 \frac{dr}{d\lambda} \frac{d\theta}{d\lambda} \mu^2 \right|, \right.$$

$$\left. \left| r^2 \frac{d\theta}{d\lambda} \dot{\mu} + r^2 \frac{d^2\theta}{d\lambda^2} \mu^2 + 2r \frac{dr}{d\lambda} \frac{d\theta}{d\lambda} \mu^2 \right| \right\} \|\Delta I_N\|$$

$$|\delta u_r| \leq \max \left\{ \left| \frac{d\theta}{d\lambda} \dot{\mu} + \frac{d^2\theta}{d\lambda^2} \mu^2 \right|, \left| \left( \frac{d\theta}{d\lambda} \right)^2 \mu^2 \right|, \right.$$

$$\left. \left| \frac{dr}{d\lambda} \dot{\mu} + \frac{d^2r}{d\lambda^2} \mu^2 - r \left( \frac{d\theta}{d\lambda} \right)^2 \mu^2 \right| \right\} \|\Delta I_N\|.$$

The joint torques that can be applied to the PACS arm are limited by saturation of the drive motors, which gives a constant torque or force limit for each joint. In addition, there are limits on the voltages which can be applied to the motors, so we need to know how the errors in the joint torques translate into errors in the motor voltages. It will be assumed that the back-EMF constant, winding resistance,

and voltage source resistance are known exactly, though this is not necessary. Since for a given speed voltage is a linear function of torque, i.e.,  $V_i = A_i u_i + B_i$ , the change in voltage will be  $\delta V_i = A_i \delta u_i$ . These changes in voltage can then be added to the nominal voltage and tested against the motor voltage limits in much the same way that the torques are checked against the motor torque saturation limits.

The perturbation to the nominal dynamics of the manipulator will be caused by placing a cube with edges of length  $L$  and uniform mass density  $\rho$  in the gripper of the robot, with its center of mass coincident with the origin of the end effector coordinate system. The pseudo-inertia of this cube is

$$\Delta I_3 = \begin{bmatrix} \frac{1}{12} \rho L^5 & 0 & 0 & 0 \\ 0 & \frac{1}{12} \rho L^5 & 0 & 0 \\ 0 & 0 & \frac{1}{12} \rho L^5 & 0 \\ 0 & 0 & 0 & \rho L^3 \end{bmatrix}.$$

The norm of this "error" is

$$\|\Delta I_3\| = \rho L^3 + \frac{1}{4} \rho L^5.$$

The maximum torque error for a given range of pseudo-inertia errors occurs when the error bound  $E$  is precisely equal to the norm of the actual pseudo-inertia error. Therefore the most stringent test of the results of the previous section is to use a tight error bound, i.e.,

$$E = \|\Delta I_3\|.$$

This has been done for a cube with sides of 5 centimeters and densities of 0, 6, 12, 18, 24, and 30 grams/cc. The path traversed is a straight line from the (Cartesian) point (0.7,0.7,0.1) to (0.4,-0.4,0.4). For comparison, the true optimal solution has been calculated, using the actual dynamics (including the effects of the cube in the gripper). The results are summarized in Tables 6.3.1 through 6.3.5. Table 6.3.1 gives traversal times for the true optimal solution and for the case in which errors are included. The "percent difference" column gives the percentage by which the true optimal traversal time is exceeded. Tables 6.3.2 and 6.3.3 give minimum and maximum voltages, respectively. The actual and nominal values are both computed for the "nominal" trajectory, i.e., the trajectory which is calculated with errors included. The actual voltages are those required to move the robot with the cube in the gripper, while the nominal values are those which are required without the cube (i.e., with the nominal payload.) The minimum and maximum voltages available are -40 and 40 volts, and it is easily seen that these limits are not exceeded for any joint or for either payload. Tables 6.3.4 and 6.3.5 give the minimum and maximum torques or forces for each joint. The torque or force limits are given at the head of the column for the appropriate joint; again, the limits are not exceeded.

The phase plane (  $\lambda$  vs.  $\mu$  ) plot and motor voltage vs. time plot for the zero-density case are shown in Figures 6.3.1a and 6.3.1b. Since the error is zero in this case, the results are exact. For a density of 12 grams/cc., the optimal and nominal (i.e., with errors included) phase plane plots are shown in Figure 6.3.2a. Figure 6.3.2b gives joint positions vs. time;  $z$  and  $r$  are in meters,  $\theta$  in radians. Figures 6.3.2c through 6.3.2e give nominal and actual motor voltages required to drive the robot along the nominal trajectory for the  $z$ ,  $\theta$ , and  $r$  joints respectively. Figures 6.3.2f



through 6.3.2h give the nominal and actual torques. (The nominal torques/voltages are those which would be required if the actual payload were identical to the nominal payload. The actual torques are the torques required to keep the robot with the perturbed payload on the nominal trajectory.) Figures 6.3.3a through 6.3.3h show the same plots for a density of 24 grams/cc.

It was noted above that none of the joint torque or voltage constraints was violated. However, the minimum voltage for the  $r$  joint at one point meets the lower voltage limit. This indicates that the trajectory which is generated when payload errors are included is indeed the fastest possible trajectory for the given range of possible payloads; for this particular point, the worst-case payload happens to have the same characteristics as the actual payload. A larger payload would have resulted in violation of a voltage constraint.

Another point to consider is the relationship between the nominal and optimal phase trajectories. It is expected that the nominal phase trajectory will be lower than the optimal trajectory; a nominal trajectory which was higher than the optimal one would lead to a contradiction of the optimality of the optimal trajectory. Also, the difference between the optimal and nominal trajectories increases as the payload error bound increases. This would be expected, since the nominal trajectory must accommodate *all* payloads within a given range; as the range of payloads increases, the worst-case errors also increase, resulting in more restrictive limits on the nominal torques, and hence slower trajectory traversal times.

Density	Time (Seconds)		percent difference
	Nominal	Optimal	
0	1.789	1.789	0
6	1.934	1.844	4.9%
12	2.076	1.898	9.4%
18	2.213	1.950	13.5%
24	2.340	2.002	16.9%
30	2.459	2.054	19.7%

Table 6.3.1. Traversal times for nominal and optimal trajectories

Minimum Voltages						
Density	$z$ joint		$\theta$ joint		$r$ joint	
	Nominal	Actual	Nominal	Actual	Nominal	Actual
0	29.86	29.86	-39.93	-39.93	-40.00	-40.00
6	30.34	30.91	-38.26	-38.52	-37.94	-40.00
12	30.54	31.67	-33.05	-33.51	-36.12	-39.98
18	30.67	32.39	-24.92	-25.37	-34.47	-39.99
24	30.78	33.07	-19.94	-20.36	-33.01	-40.00
30	30.86	33.73	-16.80	-17.18	-31.70	-39.96

Table 6.3.2. Minimum required voltages for nominal and actual payloads.

Maximum Voltages						
Density	z joint		$\theta$ joint		r joint	
	Nominal	Actual	Nominal	Actual	Nominal	Actual
0	37.64	37.64	39.86	39.86	40.00	40.00
6	37.40	38.03	37.14	37.73	32.62	32.60
12	36.79	38.04	17.64	18.25	27.01	27.48
18	35.78	37.61	10.61	11.21	23.79	24.94
24	35.12	37.54	7.30	7.89	21.39	23.41
30	34.69	37.69	5.39	5.98	19.51	22.39

Table 6.3.3. Maximum required voltages for nominal and actual payloads.

Minimum Torques/forces						
Density	z joint (Newtons)		$\theta$ joint (Newton-Meters)		r joint (Newtons)	
	Limit = -629 Nt.		Limit = -170 Nt.-M.		Limit = -15.7 Nt.	
	Nominal	Actual	Nominal	Actual	Nominal	Actual
0	333.52	333.52	-112.29	-112.29	-9.99	-9.99
6	335.91	342.20	-104.78	-105.71	-9.47	-9.99
12	363.02	376.62	-87.70	-89.26	-9.02	-9.99
18	373.03	394.00	-62.91	-64.43	-8.61	-9.99
24	377.84	406.17	-48.15	-49.55	-8.24	-9.99
30	380.11	415.72	-39.14	-40.43	-7.92	-9.98

Table 6.3.4. Minimum required torques/forces for nominal and actual payloads.

Maximum Torques/forces						
Density	z joint (Newtons)		$\theta$ joint (Newton-Meters)		r joint (Newtons)	
	Limit = 629 Nt.		Limit = 170 Nt.-M.		Limit = 15.7 Nt.	
	Nominal	Actual	Nominal	Actual	Nominal	Actual
0	421.31	421.31	161.72	161.72	10.03	10.03
6	418.68	426.52	150.39	152.36	8.15	8.18
12	414.50	430.04	78.43	80.48	6.76	6.89
18	406.68	429.55	51.39	53.41	5.96	6.25
24	402.17	432.31	38.18	40.19	5.36	5.87
30	400.42	437.96	30.20	32.20	4.89	5.61

Table 6.3.5. Maximum required torques/forces for nominal and actual payloads.

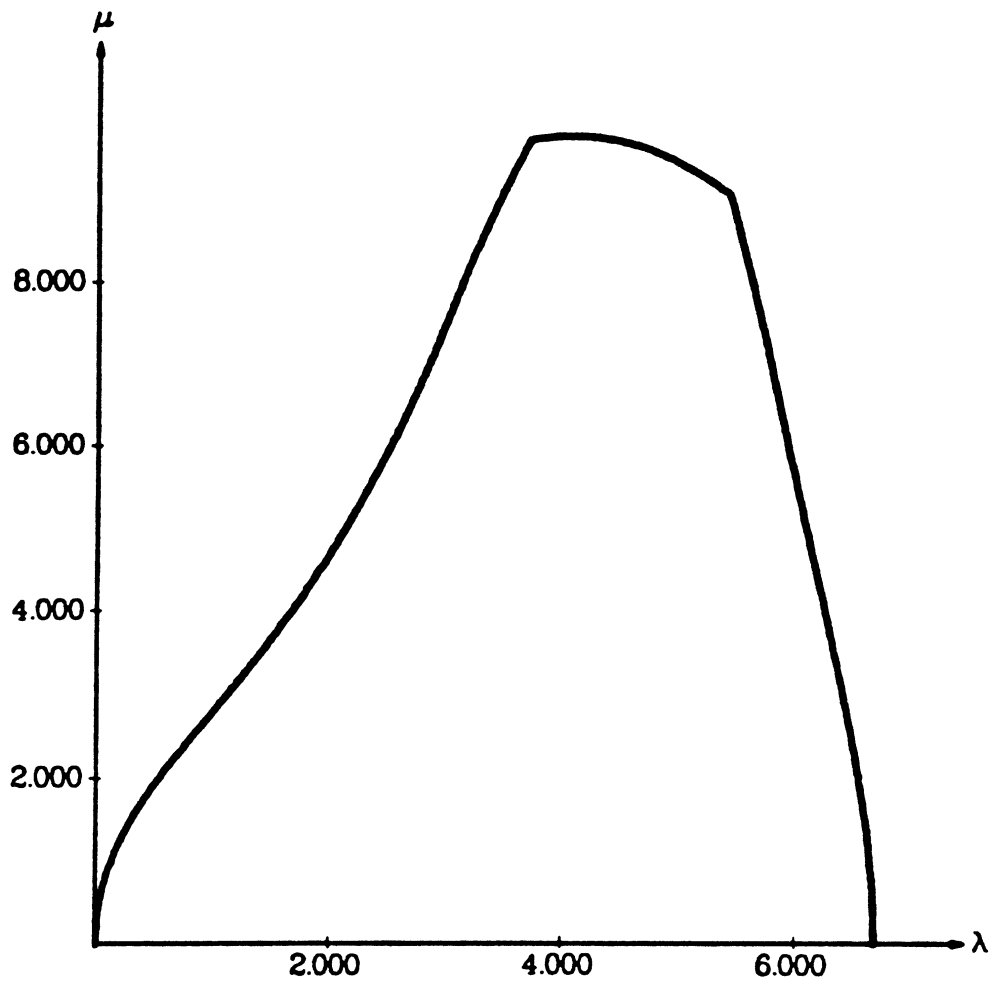


Figure 6.3.1a. Phase plane plot for zero error.

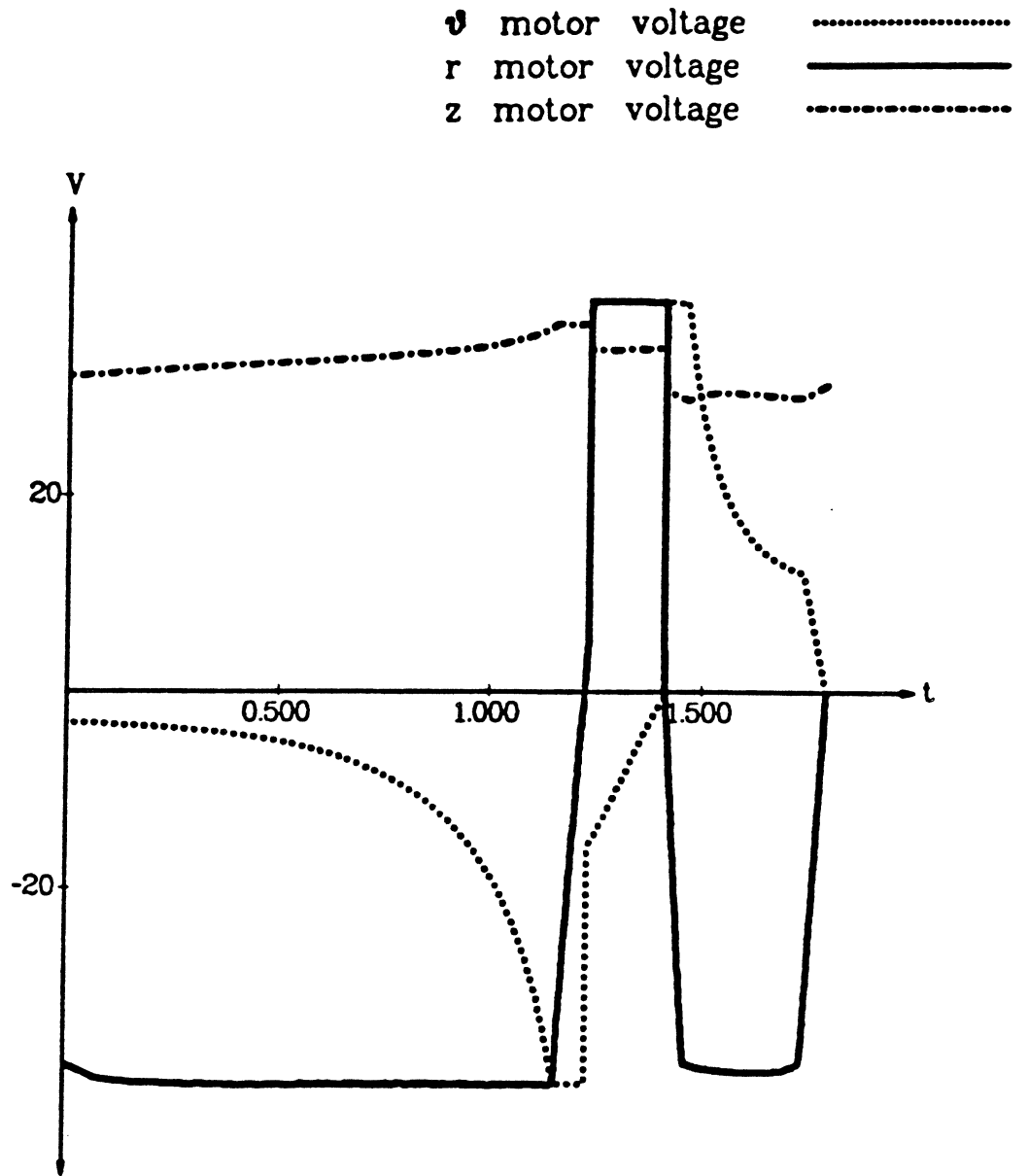


Figure 6.3.1b. Voltage vs. time for zero error.

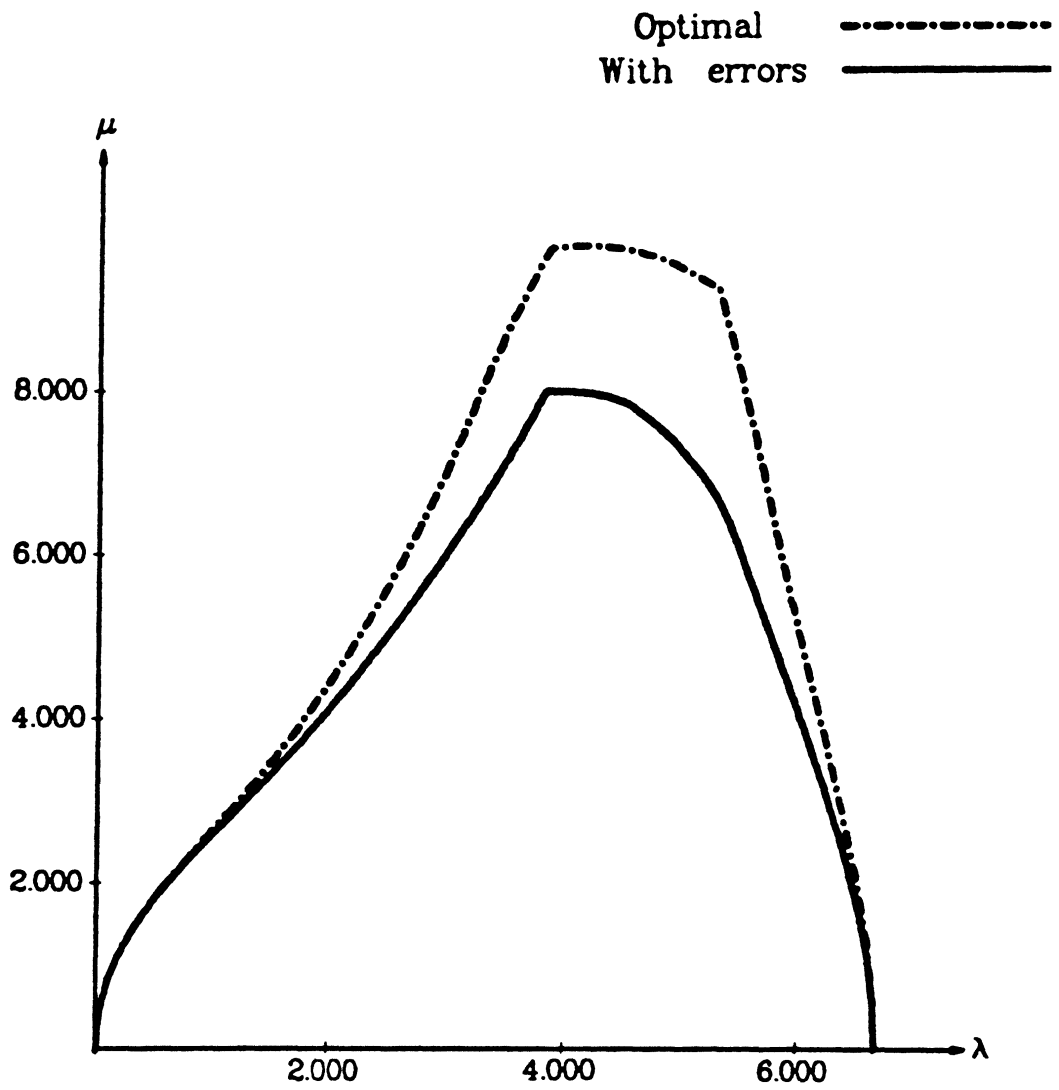


Figure 6.3.2a. Phase plane plot for density 12 g./cc.

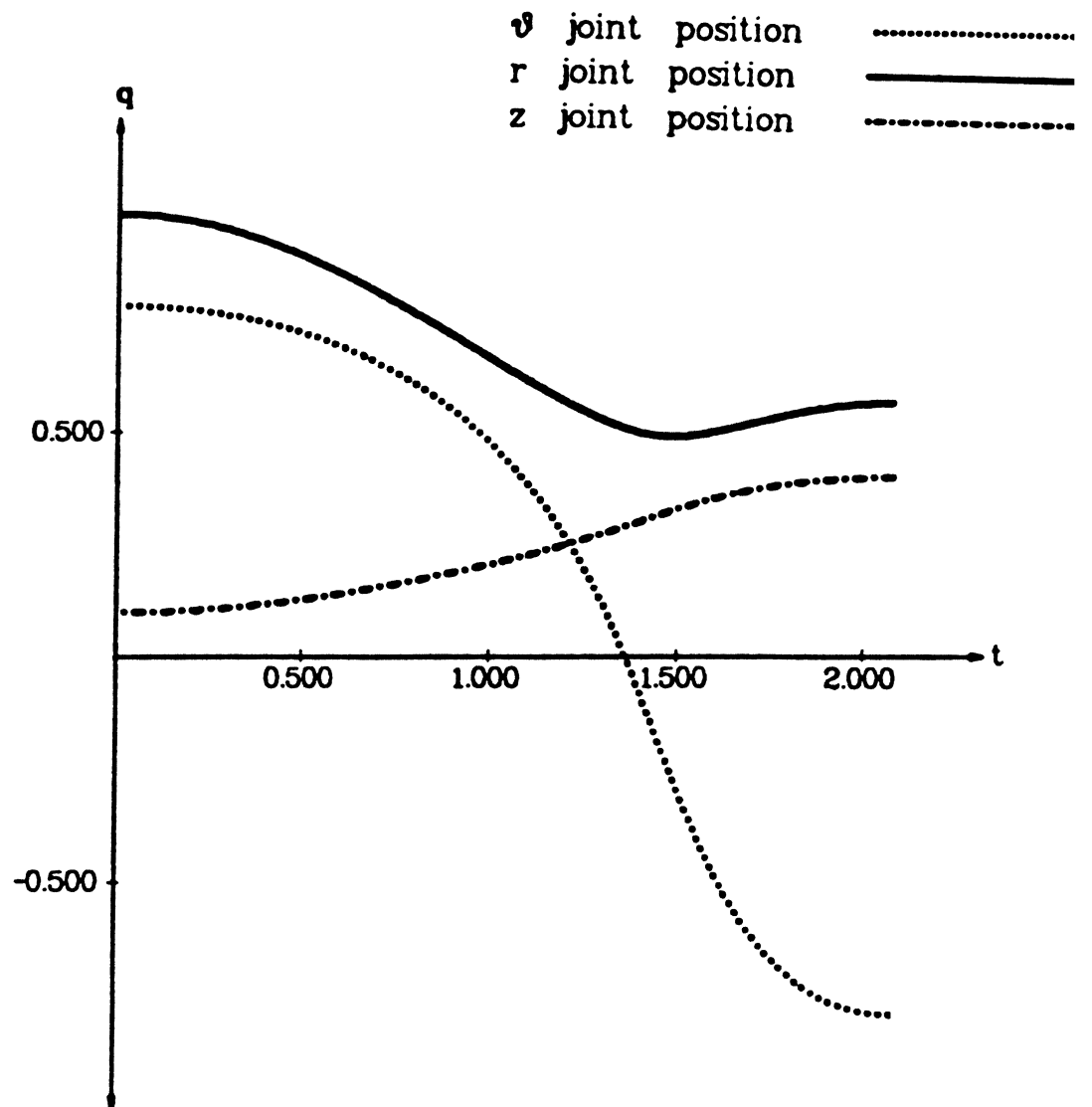


Figure 6.3.2b. Joint position vs. time, 12 g./cc.



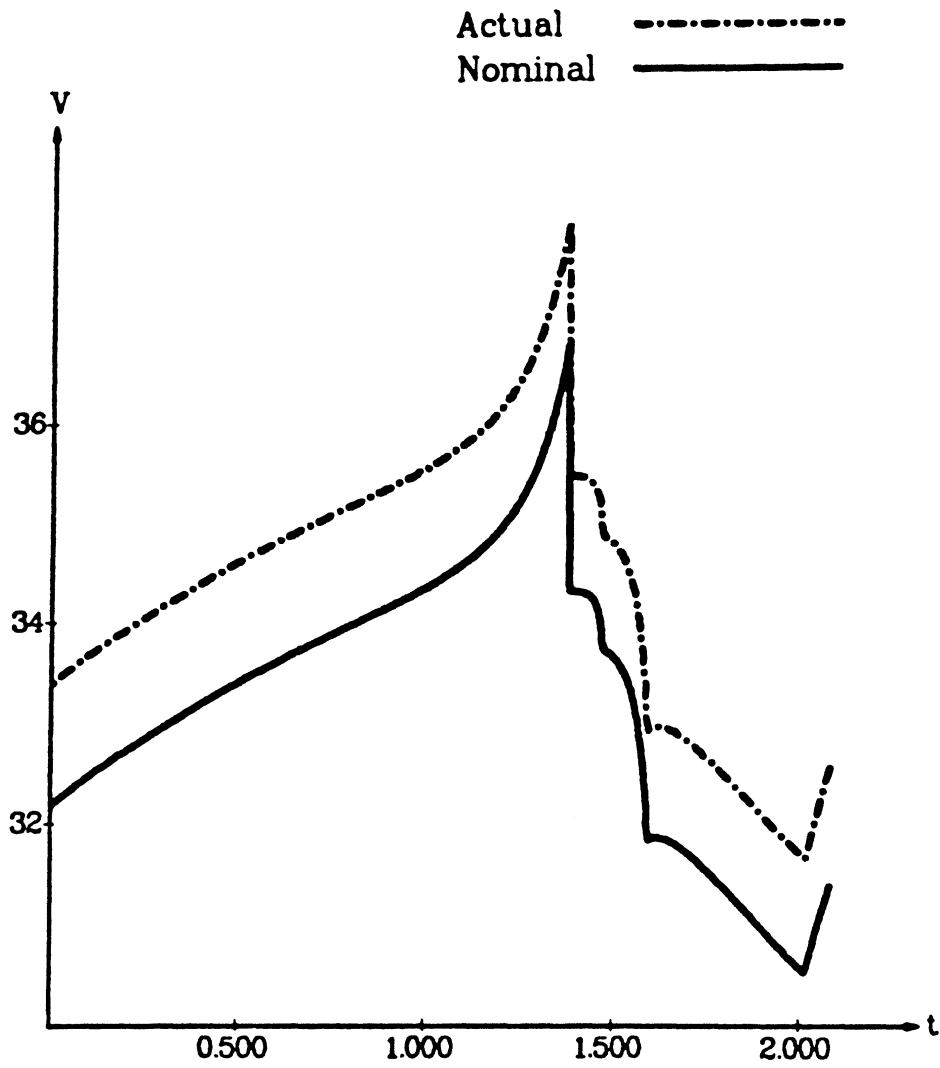


Figure 6.3.2c. Nominal and actual motor voltages, z joint, 12 g./cc.

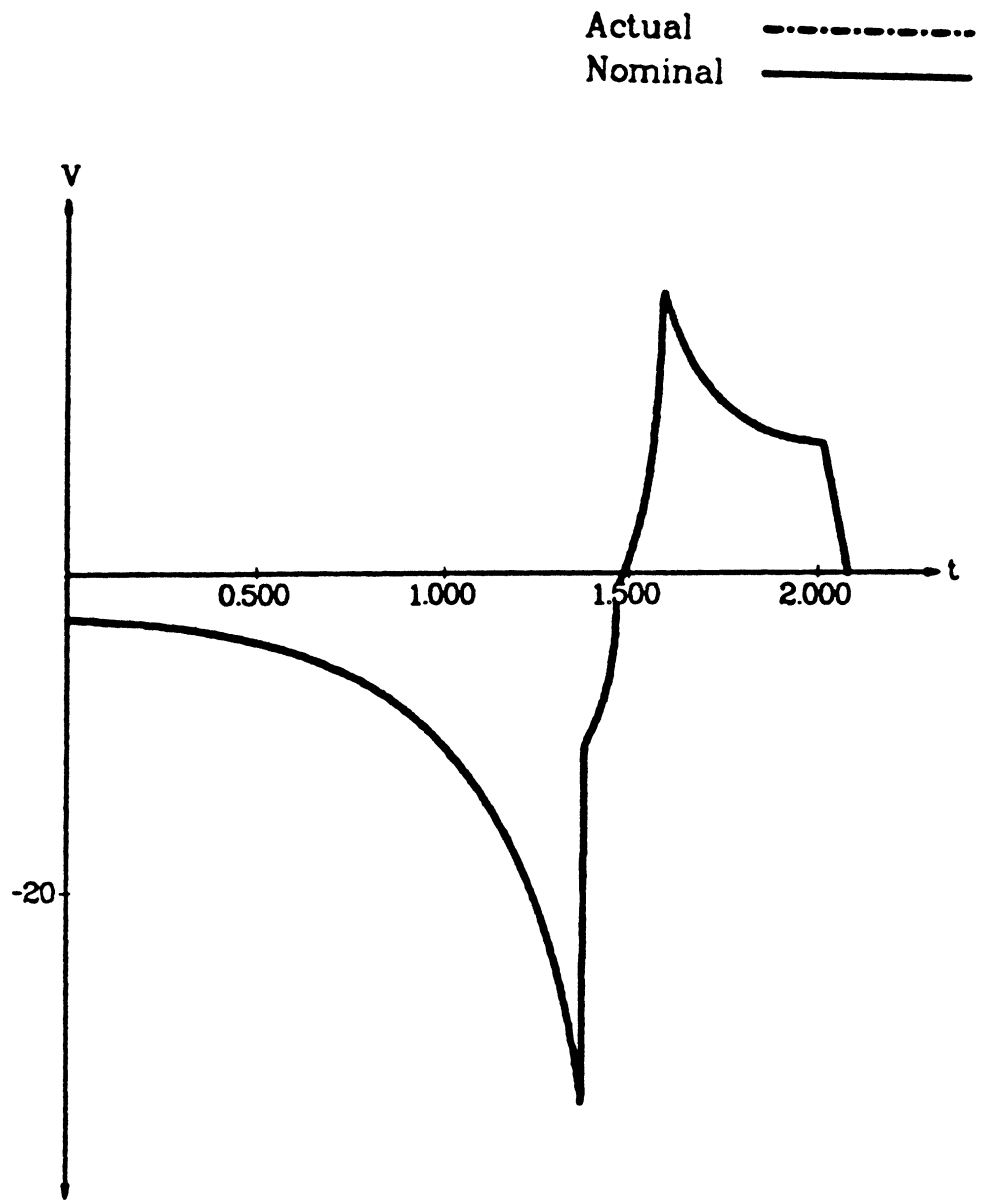


Figure 6.3.2d. Nominal and actual motor voltages,  $\theta$  joint, 12 g./cc.

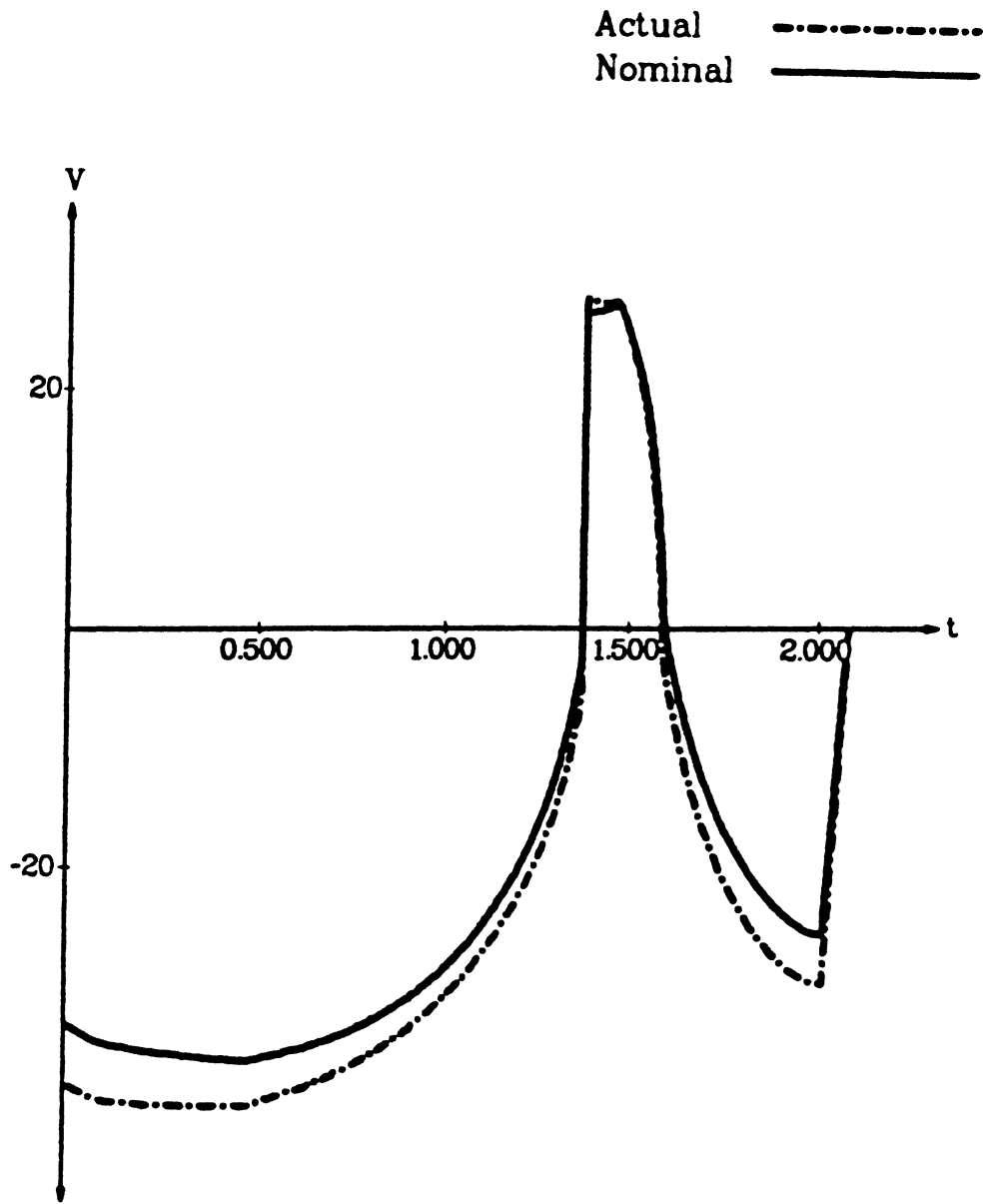


Figure 6.3.2e. Nominal and actual motor voltages,  $r$  joint, 12 g./cc.

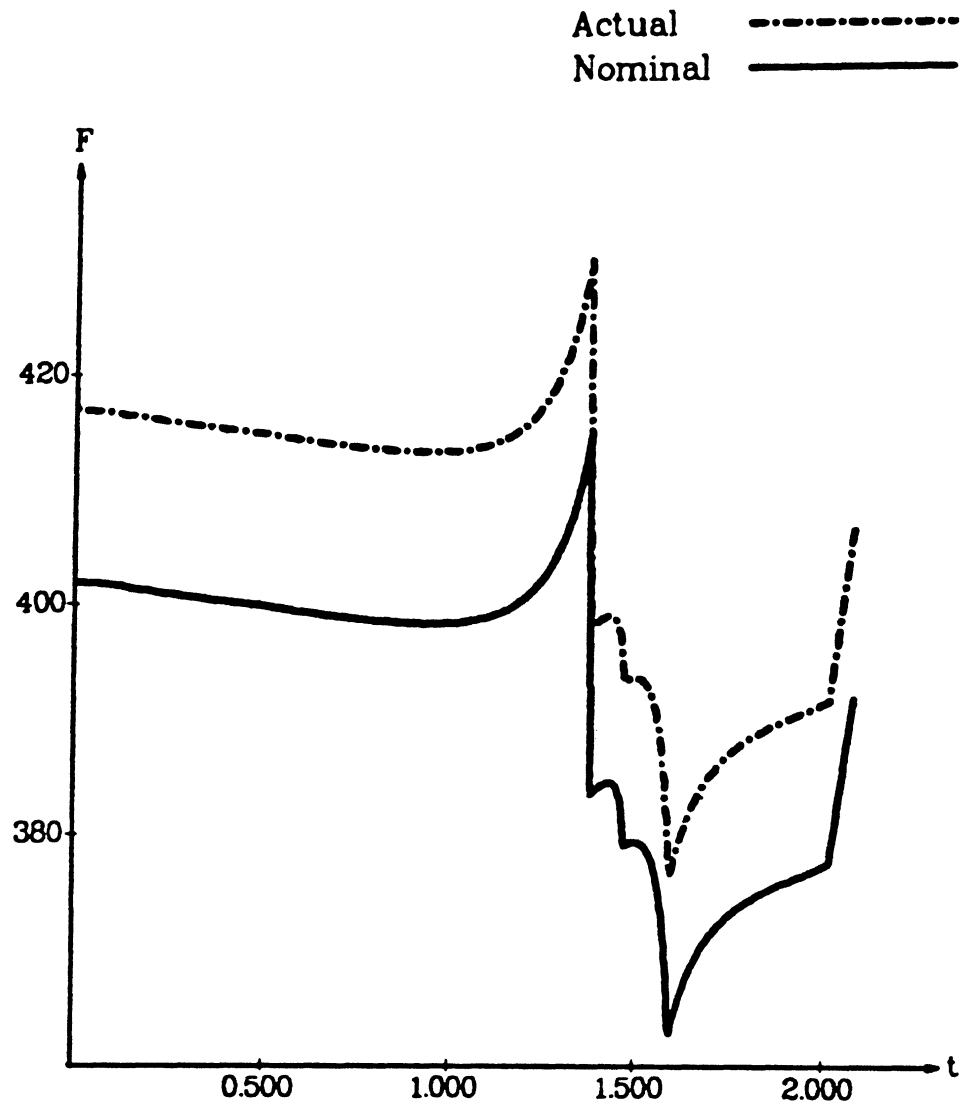


Figure 6.3.2f. Nominal and actual joint forces, z joint, 12 g./cc.

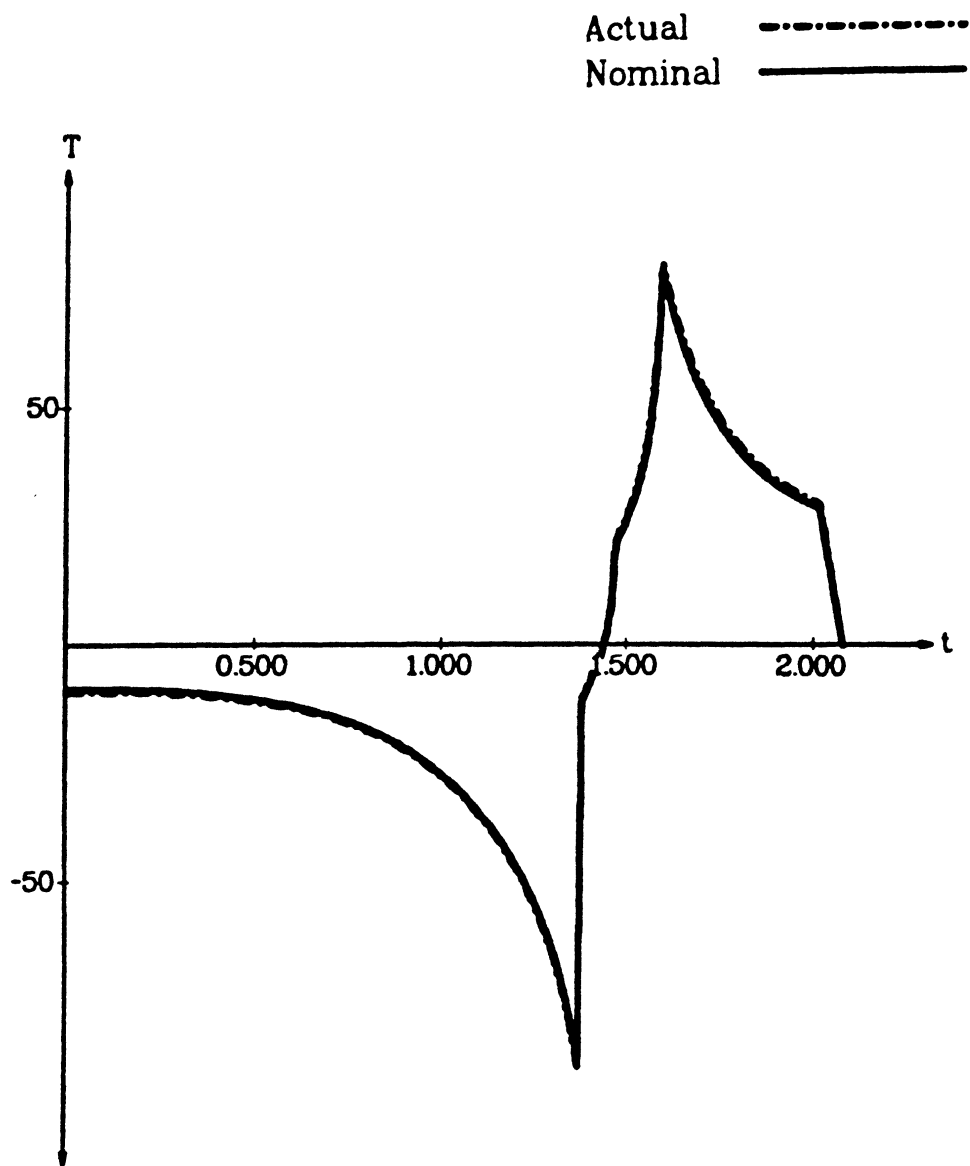


Figure 6.3.2g. Nominal and actual joint torques,  $\theta$  joint, 12 g./cc.

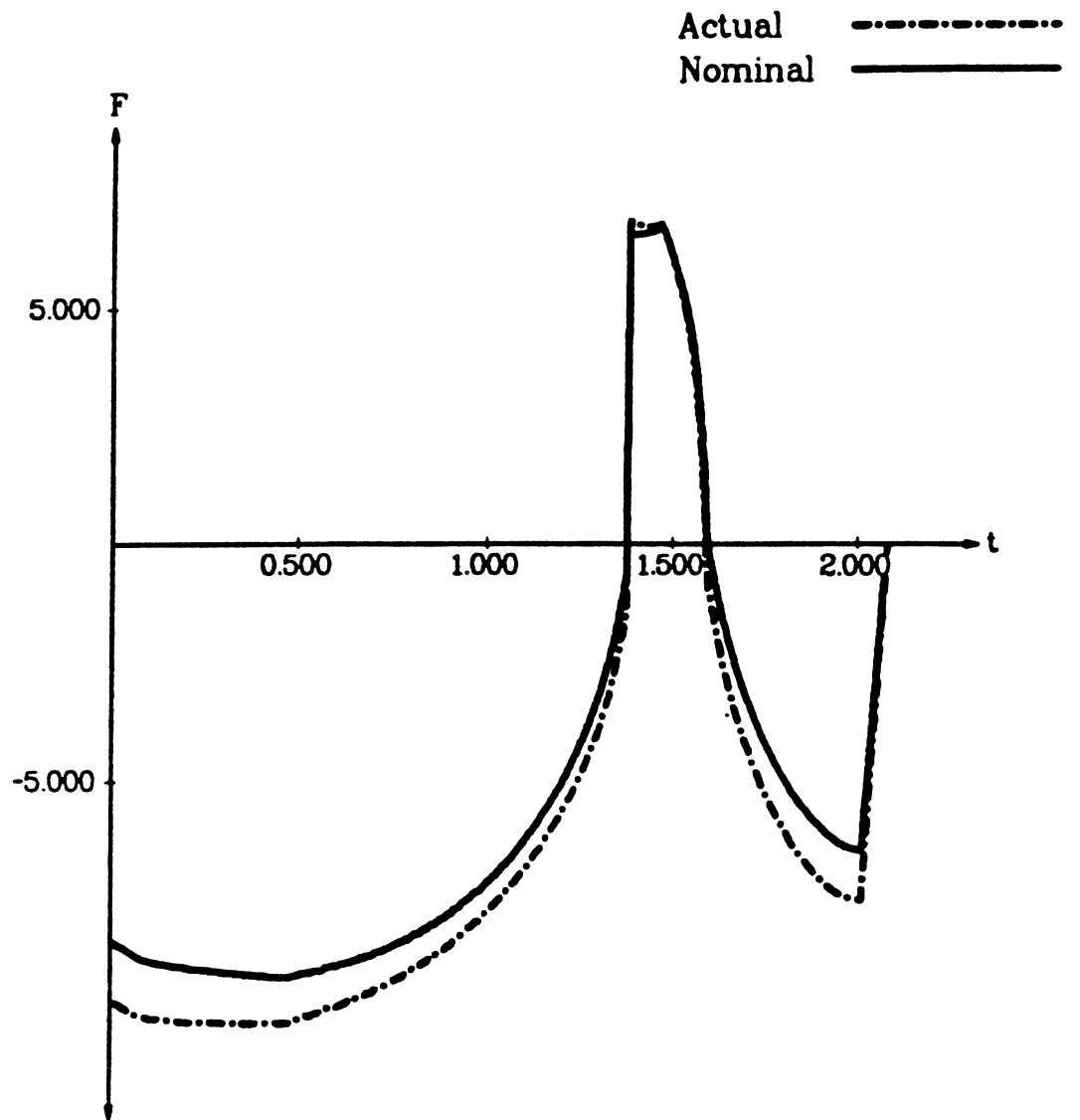


Figure 6.3.2h. Nominal and actual joint forces,  $r$  joint, 12 g./cc.

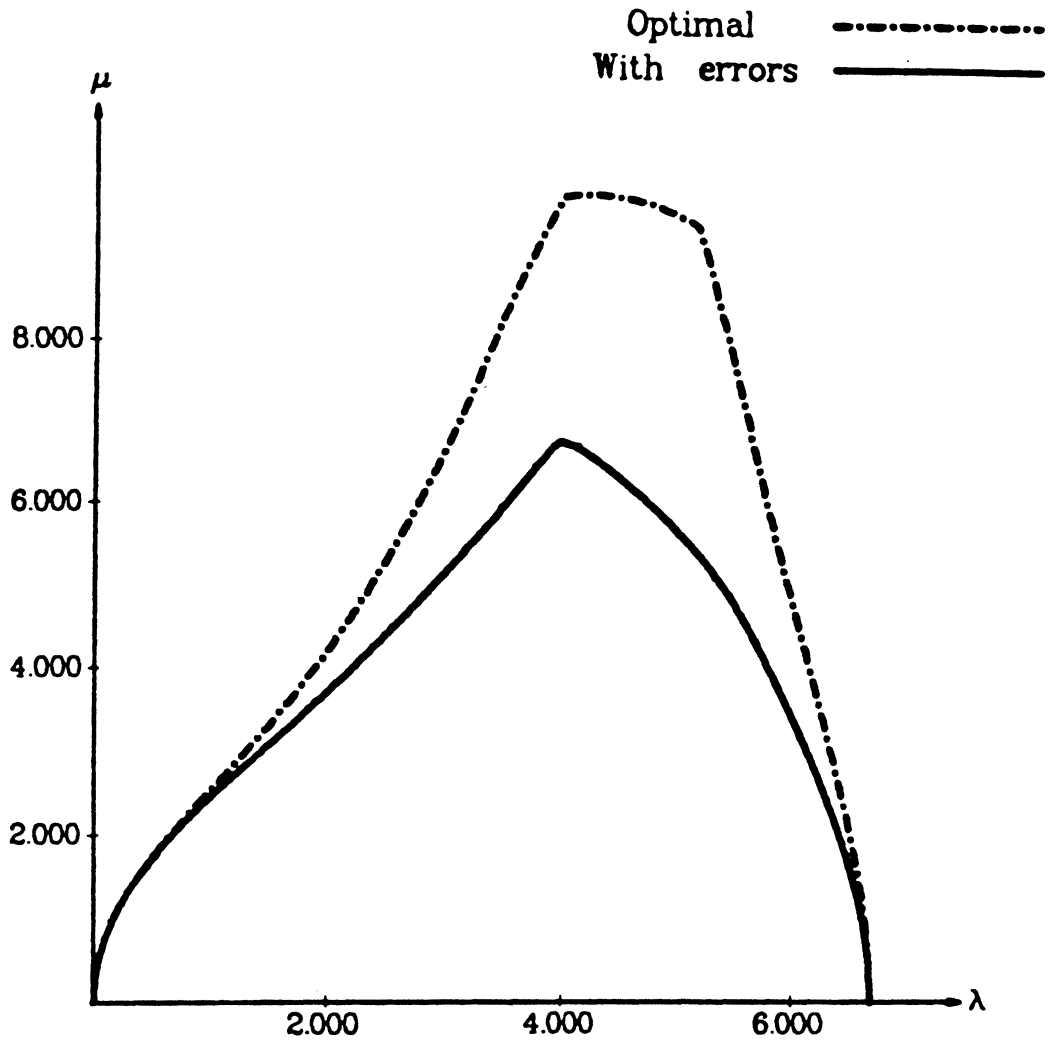


Figure 6.3.3a. Phase plane plot for density 24 g./cc.

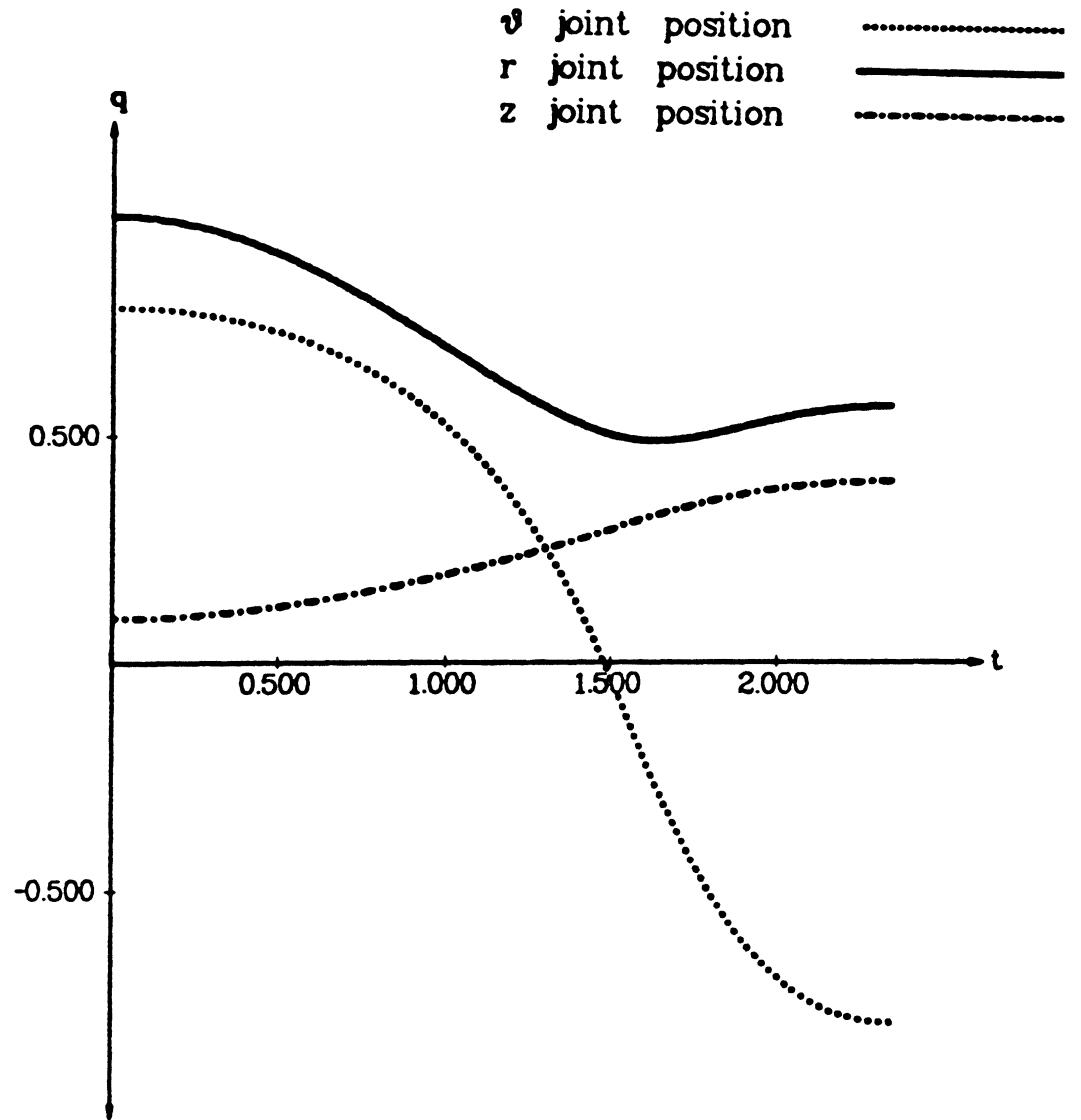


Figure 6.3.3b. Joint position vs. time, 24 g./cc.



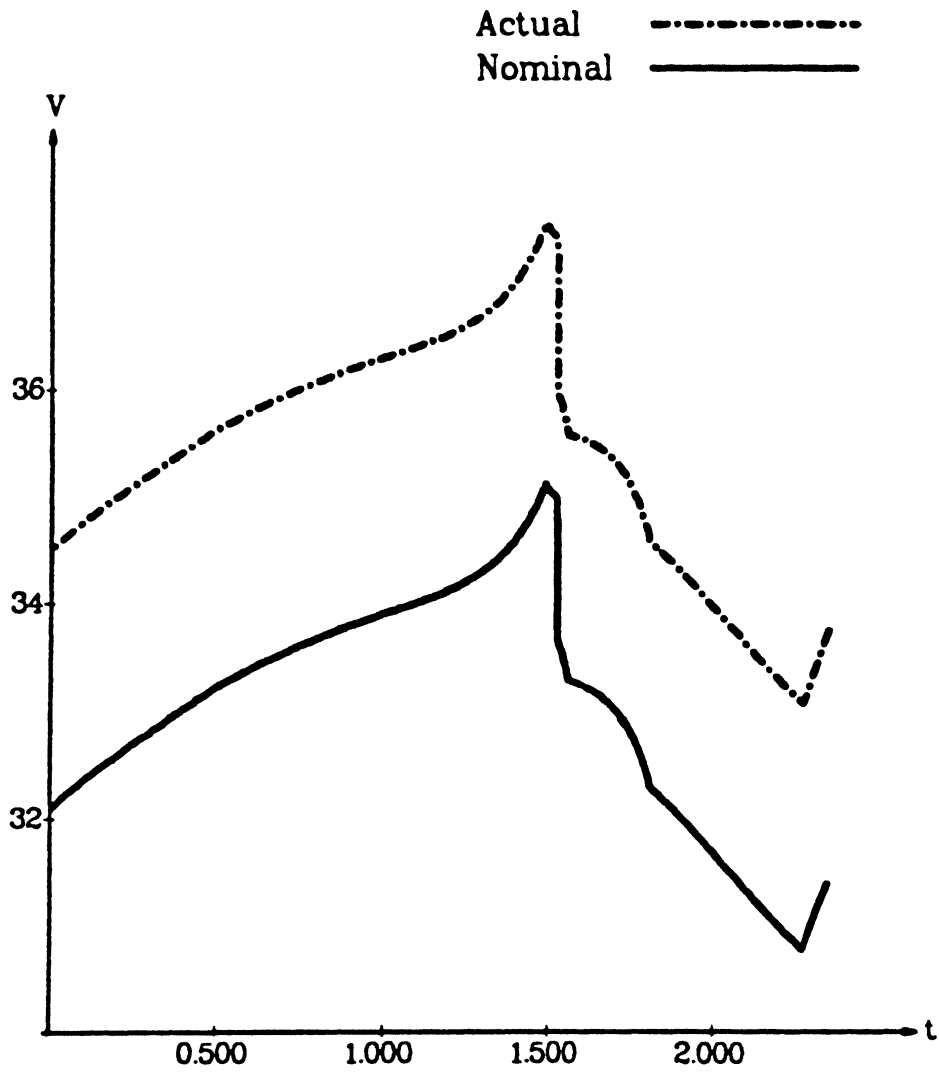


Figure 6.3.3c. Nominal and actual motor voltages, z joint, 24 g./cc.

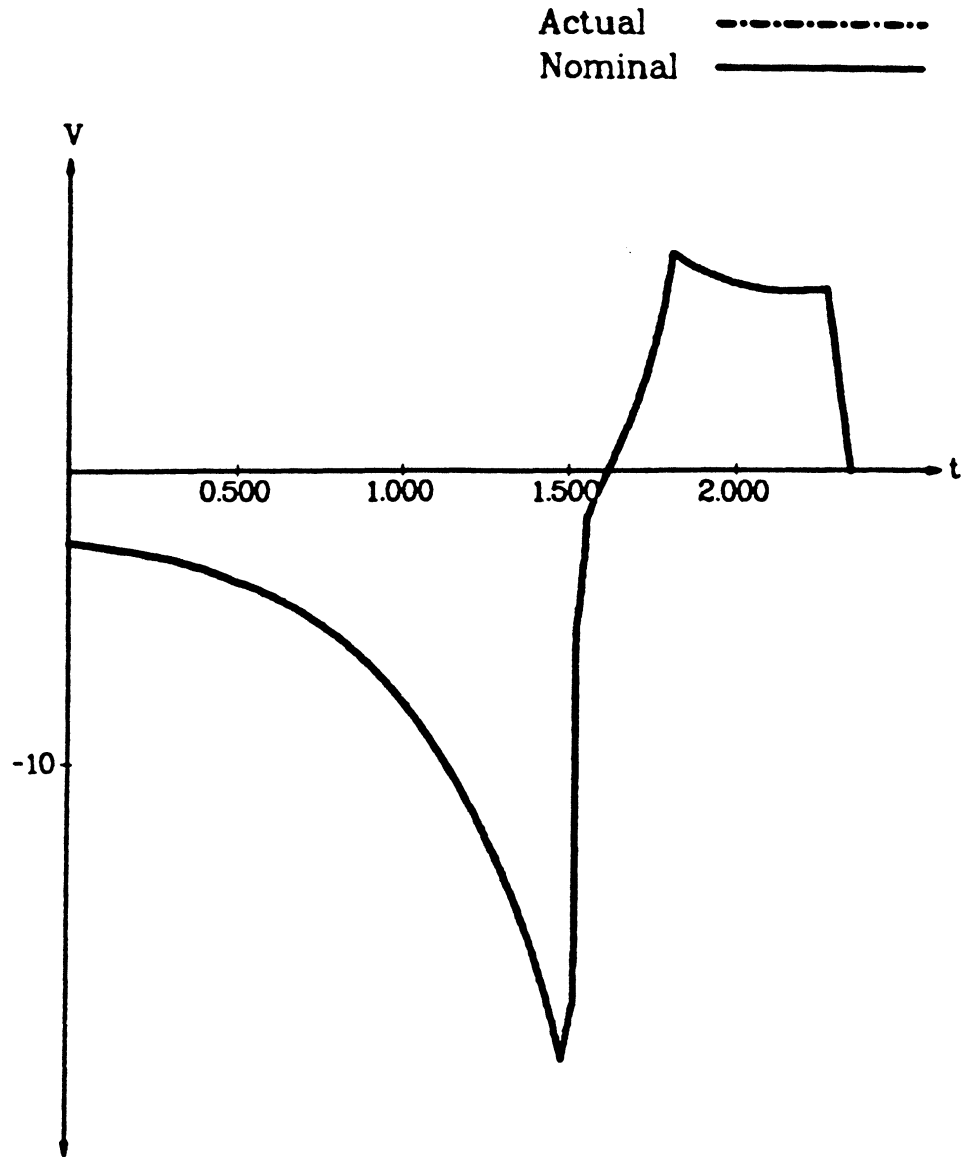


Figure 6.3.3d. Nominal and actual motor voltages,  $\theta$  joint, 24 g./cc.

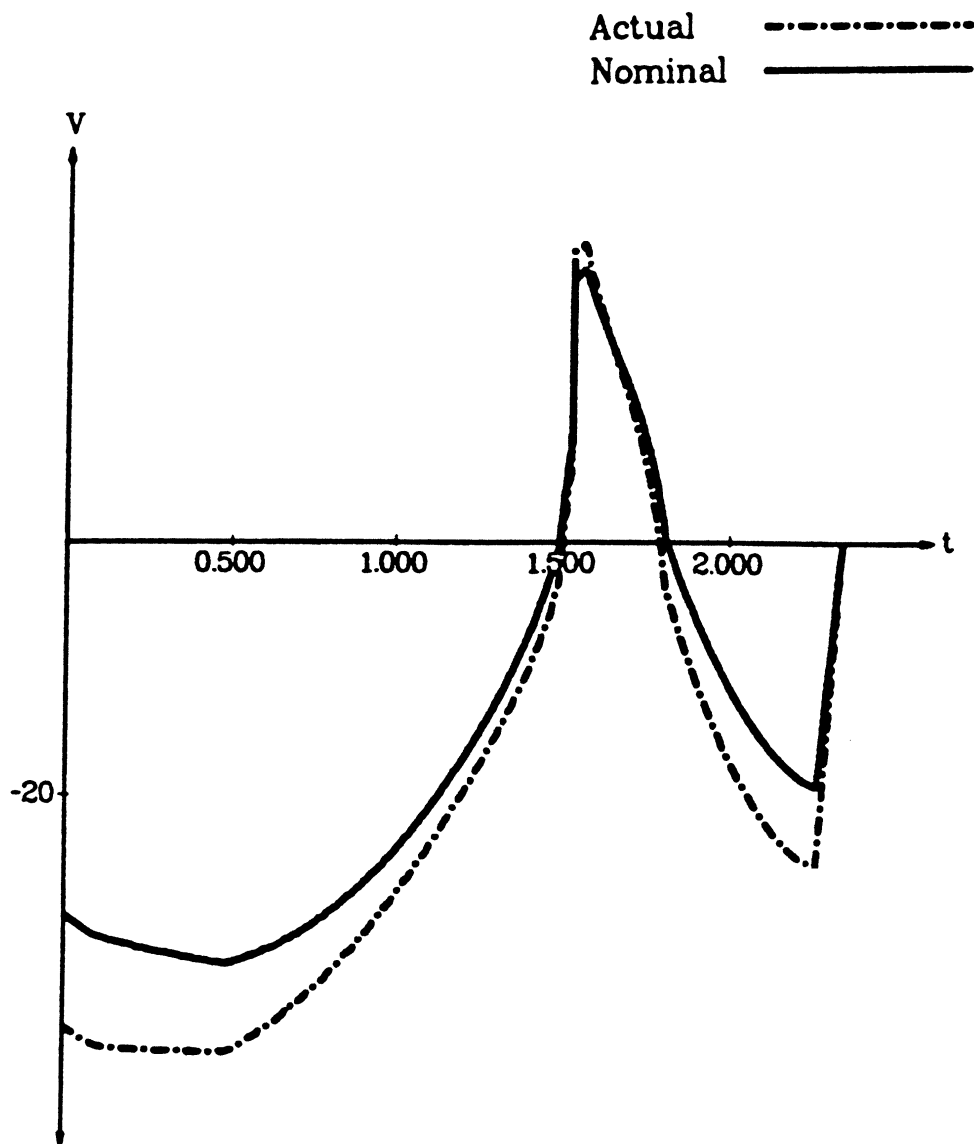


Figure 6.3.3e. Nominal and actual motor voltages,  $r$  joint, 24 g./cc.

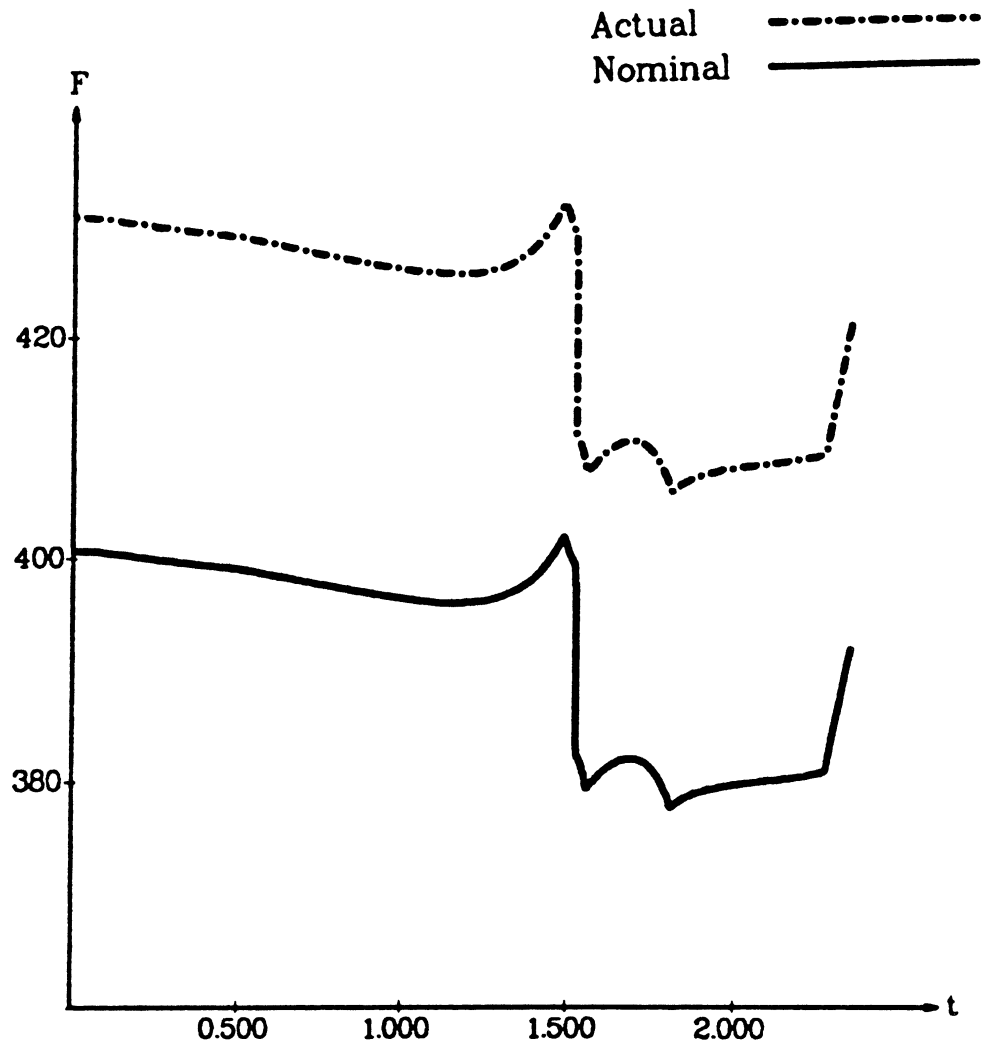


Figure 6.3.3f. Nominal and actual joint forces, z joint, 24 g./cc.

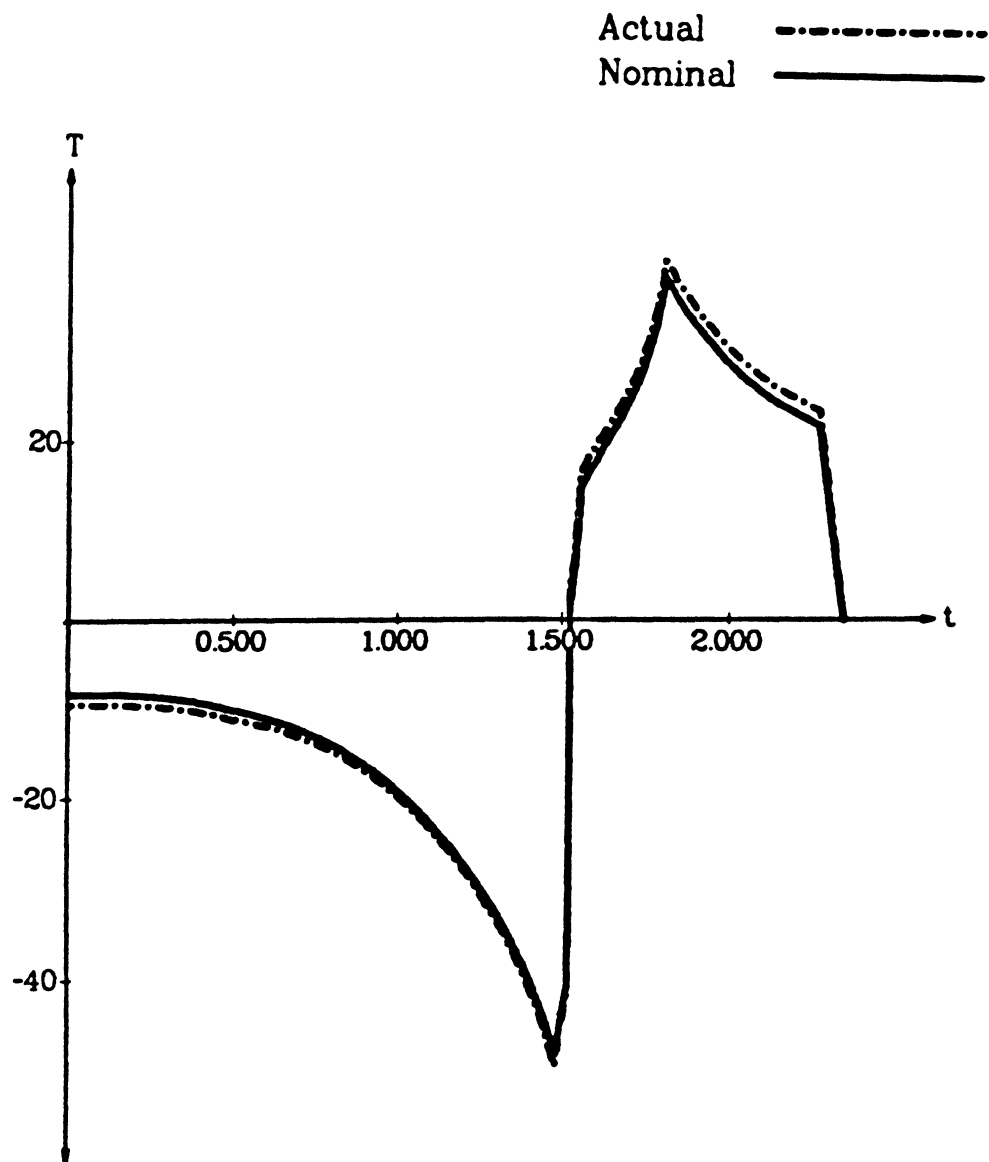


Figure 6.3.3g. Nominal and actual joint torques,  $\theta$  joint, 24 g./cc.

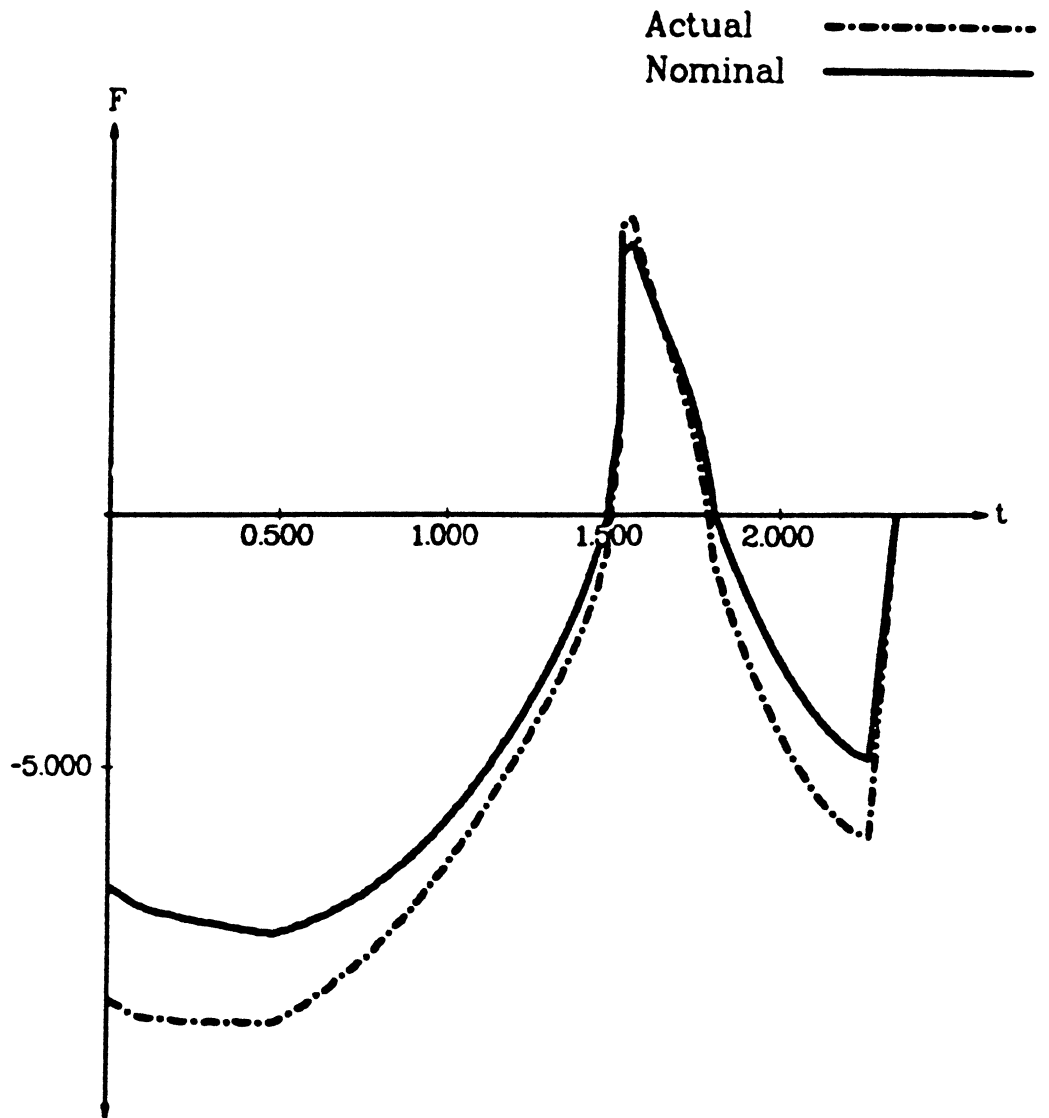


Figure 6.3.3h. Nominal and actual joint forces,  $r$  joint, 24 g./cc.

## CHAPTER 7

### AUTOMATIC GENERATION OF TRAJECTORY PLANNERS

As can be seen from the results presented in the previous chapters, writing trajectory planning programs can be a laborious task. Before actually writing the program, kinematic and dynamic equations must be derived, and actuator constraints must be described. In addition, other mundane details, such as what data structures to use, must be considered. The aim of this chapter is to describe the process of trajectory planner generation in detail, and present sufficient guidelines so that the entire process can be automated.

There are two major reasons for automating the trajectory planner generation process. First, generating a trajectory planner without machine assistance consumes much expensive human labor, even if the task is performed only occasionally. This is due in large part to the perils of hand calculation; deriving dynamic equations by hand is slow and error-prone. Second, writing a trajectory planner is time-consuming. To understand why this second factor is important, consider the use of trajectory planners as robot design aids. If trajectory planners can be generated quickly, then a hypothetical robot design can be tested easily and accurately; the robot can be pushed to its working limits, and the designer can then check to see, for example, if any joints are under-powered. It also makes the effects of design changes

easy to evaluate, since a new trajectory planner can be generated quickly, and a new set of tests can be run.

In the remainder of this chapter, the major components of a trajectory planning system are described. This description includes both functional descriptions of the components and suggestions regarding implementation details such as the choice of data structures. The generation of these components can then be reduced to algorithmic form suitable for computer implementation.

### **7.1. Trajectory Planning System Structure**

A trajectory planner is of no use in isolation; it must be part of a larger system. In a practical robot control system, the trajectory planner receives a path description from a geometric path planner, which generates the geometric path description from task descriptions. After the trajectory planner has assigned timing information to this path, it is passed on to a tracker which drives the robot in real time. However, if the trajectory planner is being used as a design aid, as suggested above, the geometric path planner, the path tracker, and the robot will not actually be present. In this case, a driver for generating geometric paths and a stub for analyzing the trajectory planner output are required. Such a system is shown schematically in Figure 7.1.1.

The precise character of the trajectory planner, the driver routine, and the stub are determined largely by the choice of representations for the input and output data and by the particular type of trajectory planner chosen. These subjects are discussed in the next section.



## 7.2. Data Structures for Representing Geometric Paths

Choosing the type of data structure used to represent geometric paths is probably the single most important decision to be made in designing a path planning system. It influences not just the design of the trajectory planner, but the design of the geometric path planner and the tracking system as well. This being the case, the geometric path representation must be considered very carefully. Several options are considered here, and their implications for the trajectory planning system design are investigated.

Three geometric path descriptions are discussed here: expression trees, splines, and arrays of points. An expression tree is a linked structure in which the internal nodes of the tree represent operators, the subtrees lying below a given node represent the operands for that node, and the leaves are irreducible expressions such as constants or variables. Such trees are equivalent to algebraic expressions. Splines are sequences of curves which are connected end-to-end. These curves usually are chosen so that they have some particular number of continuous derivatives, even at the points where the curve segments meet. For example, if cubic polynomials are used to connect the intermediate points, then it is possible to have continuous first derivatives. The second derivatives exist, but in general are discontinuous. Finally, a curve can be represented as a sequence of points. The points must be chosen so that any reasonably smooth curve which connects all the points will not deviate too much from the actual desired path

Expression trees have the advantage that their manipulation as algebraic expressions is easy. Such manipulations mimic the processes carried out by humans, and the results are complete algebraic expressions. Such manipulations are required

for deriving the robot's dynamic equations, so a formula manipulation system of some sort will be required anyway. However, expression trees have the disadvantage that introducing new types of curves for the robot to traverse may require that new types of operators be created. Also, the paths which robots are expected to traverse often are composed of connected pieces rather than single analytic curves. Some sort of conditional operator, such as a Heaviside step, is then required, and this introduces problems when the curve is differentiated; the resulting analytic expression may contain delta functions.

Cubic (or other) splines can be thought of as a restricted case of an expression tree. The expression tree will consist of the sum of many conditional expressions, where each conditional expression consists of two step functions multiplied by a cubic polynomial. For instance, the segment of a spline which passes from point 5 to point 6 might be represented as

$$s(\lambda - \lambda_5) s(\lambda_6 - \lambda) p_5(\lambda), \quad (7.2.1)$$

where

$$s(\phi) = \begin{cases} 0 & \text{if } \phi < 0 \\ 1 & \text{if } \phi \geq 0 \end{cases} \quad (7.2.2)$$

and  $p_5(\lambda)$  is a cubic polynomial in  $\lambda$ . The problems discussed above for expression trees also apply to cubic splines.

A simple array of points does not lend itself to symbolic manipulation of any sort. However, most of the calculations which the trajectory planner must carry out can be done numerically, so this is not a big disadvantage. The major disadvantage of this method is that the path is not really completely specified; the motion between

the interpolation points is undetermined. However, if the density of interpolation points is high enough, this will not matter in practice. Indeed, the perturbation and dynamic programming trajectory planners discretize the trajectory planning problem anyway, and the differential equations that need to be solved when using the phase-plane method must in practice be solved using discrete, approximate methods. The representation is very simple, and for that reason was used in most of the examples presented in the previous chapters. It is also a very easy structure to attach other information to; if the geometric path is represented as an array of points, and the points are represented as a structure or record, then time, torque, and velocity information can be attached to each point by simply adding new fields to the record. Attaching this information to a spline or an expression tree is much more difficult. For these reasons, an array of points seems to be a good choice of data structures.

The data structure used in the examples in this thesis is shown in Figure 7.2.1a. It is a record, and it contains information which applies to the path as a whole: the maximum value of the parameter  $\lambda$ , the number of joints the robot has, and the number of interpolation points on the curve. In addition, this record contains pointers to several arrays, including an array of  $\lambda$ -values for each point on the curve. Finally, it contains a pointer to an array of "joint paths", where each joint path is an array of joint data values, one array element per point. It is in these arrays that the information pertaining to the individual joints, such as the dynamic coefficients  $M_i$ ,  $Q_i$ ,  $R_i$ , and  $S_i$ , the parametric derivatives  $\frac{d q^i}{d \lambda}$  and  $\frac{d^2 q^i}{d \lambda^2}$ , and the joint torques  $u_i$  and motor voltages  $V_i$ , are stored. A typical structure for these values (the  $D_{ij}$  in Figure 7.2.1a) is shown in Figure 7.2.1b.

The trajectory planning process can be expressed entirely in terms of operations on this data structure. The input to the trajectory planner is a sequence of points, and the ultimate goal of the trajectory planner is to fill in the values of  $\mu$ ,  $\dot{\mu}$ ,  $t$ , etc. at each point on the curve. The intermediate steps are to compute the values of the parametric derivatives  $\frac{d\mathbf{q}^i}{d\lambda}$  and  $\frac{d^2\mathbf{q}^i}{d\lambda^2}$ , calculate the dynamic coefficients  $M_i$ ,  $Q_i$ ,  $R_i$ , and  $S_i$ , and finally apply a trajectory planning algorithm which generates  $\mu$ ,  $\dot{\mu}$ ,  $t$ , and  $u$ .

### 7.3. Selection of a Trajectory Planning Method

The choice of geometric path representations influences the structure of the entire path planning system, from the geometric path planner down through the tracking system. The choice of trajectory planning techniques has more localized effects, but the effects on the implementation of the trajectory planner, which are of primary concern here, are major. Assuming that minimum-time trajectories are desired, the perturbation method should be chosen rather than dynamic programming, since it is considerably faster and can handle more general torque constraints. The phase plane method is the fastest of the three techniques described in this thesis, but cannot be generalized to handle all the kinds of torque constraints that the perturbation method can handle. The torque constraints also must be incorporated into the phase plane algorithm in several places, making changes to the torque limits more difficult in a phase-plane trajectory planner than in a perturbation trajectory planner. The perturbation method isolates the torque constraints into a single constraint function, thereby isolating the torque constraint checks from the rest of the

trajectory planner.

Because of the ease of construction of perturbation trajectory planners, and because of the natural way that they break up into modules, they would appear to be good choices for a computer-generated trajectory planner. The major components of such a trajectory planner are shown in Figure 7.3.1. The components are a derivative generator, a dynamic coefficient generator, and the trajectory planning algorithm itself. Note that the constraint function “plugs into”, or links to, the trajectory planning algorithm only, and that the trajectory planning algorithm itself is the same regardless of the torque constraints. The trajectory planning algorithm also is the same for any robot, regardless of its dynamics; everything that the trajectory planning algorithm needs to know about the robot’s dynamics is distilled into the dynamic coefficients  $M_i$ ,  $Q_i$ ,  $R_i$ , and  $S_i$ . The derivative generator can be a simple numerical procedure and is independent of the robot characteristics also, so that only the constraint function and the dynamic coefficient generator need ever be changed.

#### 7.4. Generation of Dynamic Equations

So far, very little has been said about the actual generation of the dynamic equations for robots. This is the most time-consuming, labor-intensive and error-prone part of the trajectory planner generation process, and so is the most important part to automate. Computer generation of dynamic equations may be time-consuming, but it is certain to be several orders of magnitude faster than hand calculation, and certainly will be more accurate.

Though it sometimes is possible to derive dynamic equations of simple robots relatively quickly in *ad hoc* ways, such tricks are difficult to apply systematically.

However, systematic methods do exist; see [28]. These systematic methods, combined with a system for doing symbolic algebra, such as MACSYMA [27] or REDUCE [10], and a symbolic differentiator make the generation of dynamic equations fairly straightforward. While the resulting equations may not be in simplest form, they will certainly be correct.

It is important to note that having the dynamic equations in simplest form may not always be necessary. In particular, this is the case if the trajectory planning system is to be used primarily as a tool for robot design. In that situation, the trajectory planner will probably be run several times and then thrown away as design changes are made. It is much more important that the trajectory planner be generated quickly than that it run quickly. (Essentially the same reasoning is used to justify the existence of slow optimizing and fast non-optimizing compilers for conventional computer languages; there usually is no sense in spending ten minutes to optimize a small program that will be run only once or twice.)

A schematic of the robot dynamics generation process is shown in Figure 7.4.1. The user first describes the robot's kinematics; this can be done by specifying the number of joints, whether the joint is revolute or prismatic, and the various joint offsets and twists. The resulting transformation matrices can be computed as in [28]. Once the forward kinematics of the robot are known, the user may describe the link inertias. A simple version of a dynamic equation generator may just prompt the user for the link pseudo-inertias; a more sophisticated version would compute the inertias from CAD models of the robot links. Given the kinematics and the pseudo-inertias, the inertia matrix, the Coriolis coefficient array and the gravitational forces can be calculated using the formulas in [28]. It is for these computations that a

symbolic differentiator is needed, since the kinematic transform matrices must be differentiated with respect to the joint variables.

The output of the dynamic equation generator will be arrays of expression trees representing the inertia matrix, the Coriolis coefficient array, and the gravitational force terms. These must be translated from arrays of expression trees to some form suitable for compilation by whatever high-level language compiler is chosen as an implementation language. If the target language has pointers to procedures, then an appropriate representation of, for example, the inertia matrix would be a two-dimensional array of pointers to procedures; each element of the array would be set to point to a function which would compute the appropriate inertia matrix coefficient, given the robot's current position. This requires only that the expression tree representations be translated into appropriate computer code, a very simple task.

### **7.5. Generating the Constraint Function**

Generating the constraint function for a dynamic programming or perturbation trajectory planner poses some problems which are not amenable to general solution. Actuator torque constraints may in general be quite complicated; for example, in a cable-driven robot arm, moving one drive cable may move several joints, cables will stretch, and the cables must all be kept under positive tension. Generating constraint functions for completely arbitrary actuators is obviously very difficult if not impossible. However, not every actuator which is theoretically possible, nor even every actuator which is practically realizable, need be considered. By restricting the trajectory planner generator to a few very common actuator types, most practical robot designs can be handled.

One type of actuator which is very common is the D.C. torque motor. The torque bounds for such a device are determined by the saturation limits of the motor itself and by the properties of the amplifier which drives the motor. In addition, there are limits on the time derivative of the torque which are imposed by the motor inductance and the drive amplifier voltage limits, but these constraints can often be ignored in practice. Since only a few parameters are needed to describe the motor torque limits, generating constraint functions for this case is a fairly simple process.

Other types of actuators, such as hydraulic servoes, may also be described in a simple manner provided that effects such as delays due to the finite speed of propagation of pressure waves are ignored. As long as these effects are small, the resulting constraint function should produce a useful trajectory planner.

### **7.8. Ancillary Software**

As indicated in Figure 7.1.1, the trajectory planner is not the only component of a path planning system; if the trajectory planner is to be used as a design tool, a driver for generating test data and a stub for analyzing trajectory planner output are both required.

The stub can be a very simple routine. Most of the time a graphical representation of joint speeds and forces suffices, and the generation of such output can be performed in a manner which does not depend on the kinematics or dynamics of the robot. The driver routine, on the other hand, may depend rather heavily upon the robot's kinematics. If the designer wants to evaluate the performance of the robot as it moves along a Cartesian straight line, then the inverse kinematics of the robot are required. This causes some problems, since the inverse kinematics of robot arms can-



not in general be computed in closed form, and because the inverse kinematic solutions are not always unique. However, many robots fall into a relatively small set of kinematic configurations, so that the inverse kinematics can be computed for a generic member of each kinematic class. Then the designer need only plug the precise link dimensions into a standard formula. (See [5] for an example.) While this approach may not work for certain exotic robot designs, it probably will cover the vast majority of robots seen in practice.

Another desirable feature of the driver routine is the generation of geodesics in inertia space. The traversal times for these curves are near-optimal, and so will give the robot designer an estimate of the robot's maximum capabilities. Since the differential equations of inertial geodesics are directly obtainable from the robot's dynamic equations, much of the work of writing a geodesic generator will have been done already. The equations need only be incorporated into an appropriate differential equation solver.

The proposed trajectory planner generation system is shown schematically in Figure 7.6.1. The user provides kinematics, link inertias, and actuator characteristics, and the planner generator produces a trajectory planner, a stub, and a driver. The driver allows selection of straight line paths in joint space, Cartesian straight lines, and geodesics.

### **7.7. Work Accomplished**

Most of the work performed to date on the automatic generation of trajectory planners consists of the construction of a system for manipulation of algebraic expressions and a symbolic differentiator. These routines consist of about 1400 lines

of C, with an additional 1300 lines of code for testing the differentiator. Though these utilities only constitute a portion of the trajectory planner generating system, they are very important components. The largest single component of the trajectory planner generator will most likely be the dynamic equation generator, of which the algebraic manipulation system is an integral part.

Some of the rest of the code for the trajectory planner consists of fixed modules; these portions can be taken from the code written for the numerical examples in this thesis, and used either directly or with minor modifications.

Though much work remains to be done, it is clear from the discussion presented here that the major problems involved in producing an automatic trajectory planner generator involve writing some large but well-defined pieces of code; there are no major conceptual or theoretical problems to be solved.

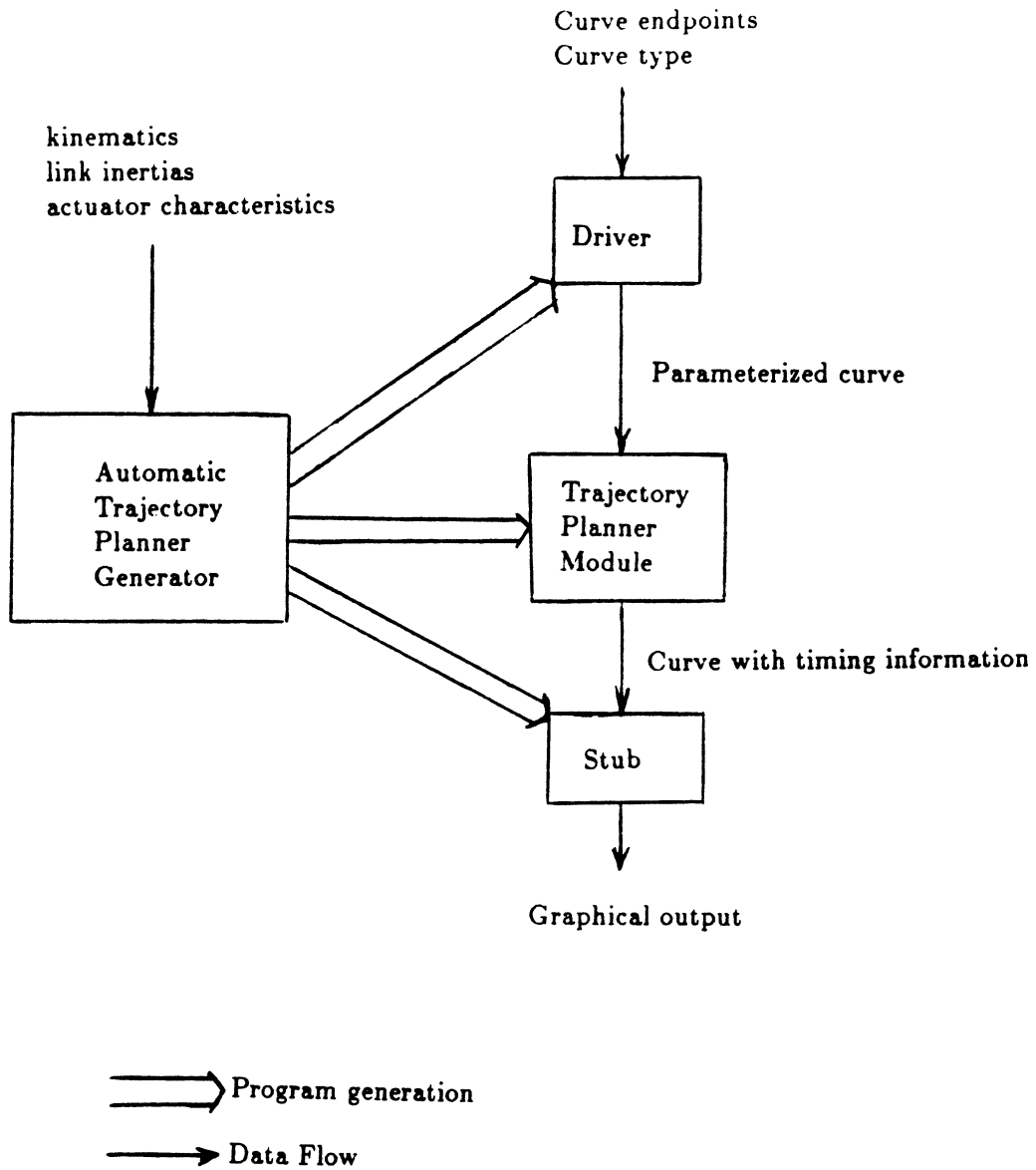
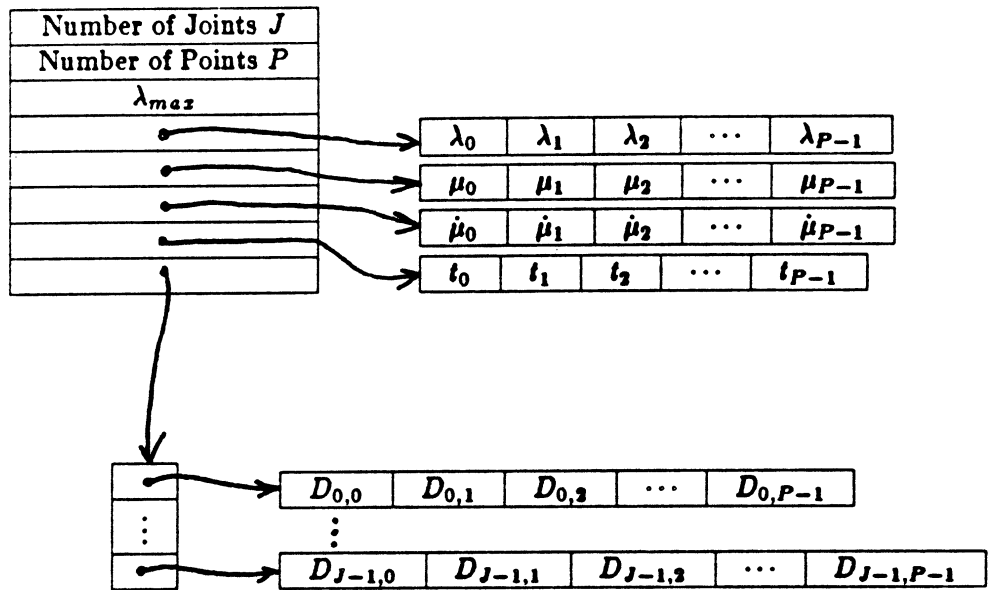


Figure 7.1.1. Overview of ATPG system



a) Path data structure

$q_m^i$	$\frac{dq_m^i}{d\lambda}$	$\frac{d^2q_m^i}{d\lambda^2}$	
$M_i(\lambda_m)$	$Q_i(\lambda_m)$	$R_i(\lambda_m)$	$S_i(\lambda_m)$
$u_m^i$	$V_m^i$		

b) The structure  $D_{i,m}$

Figure 7.2.1. Robot path data structures

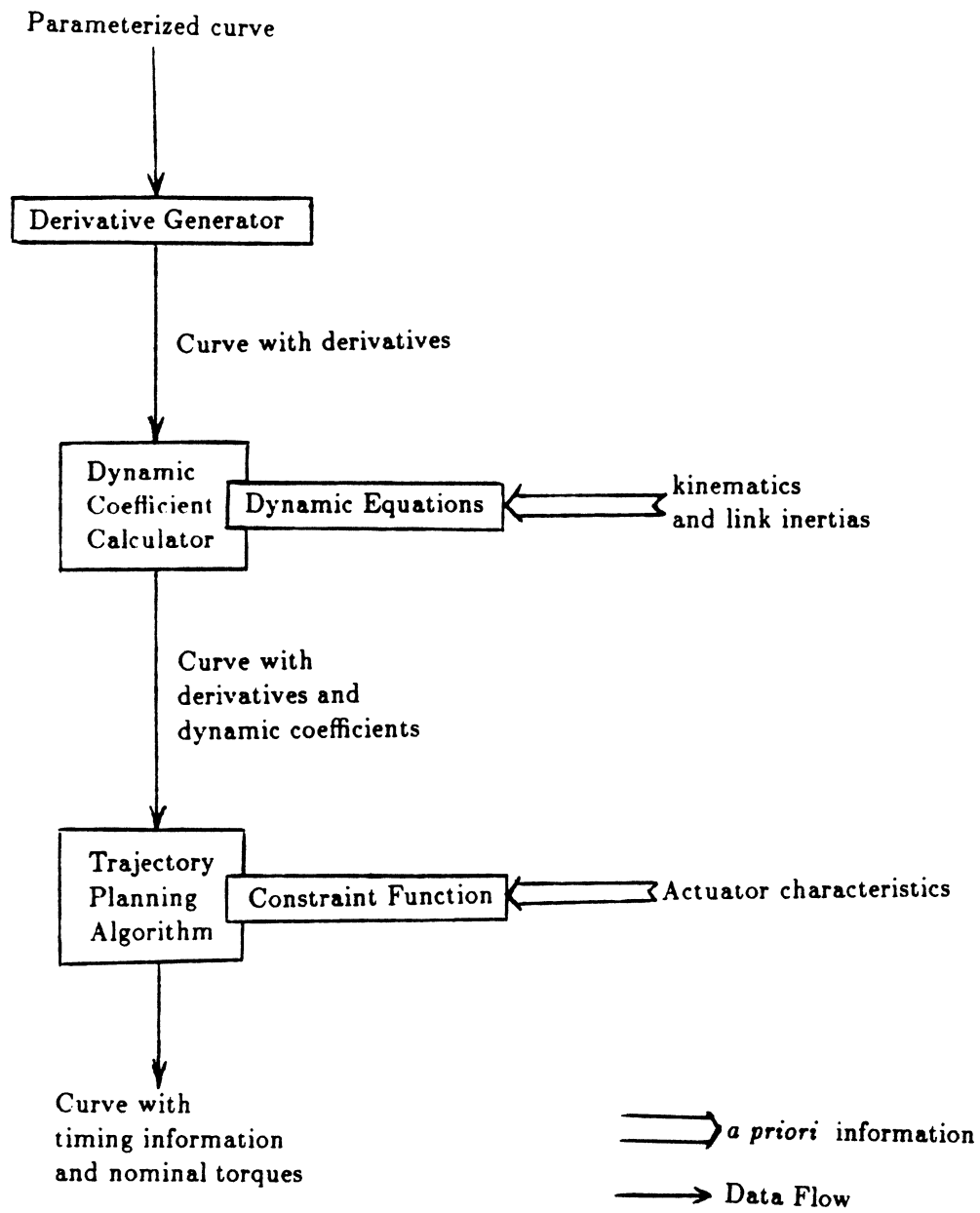


Figure 7.3.1. Trajectory planner structure

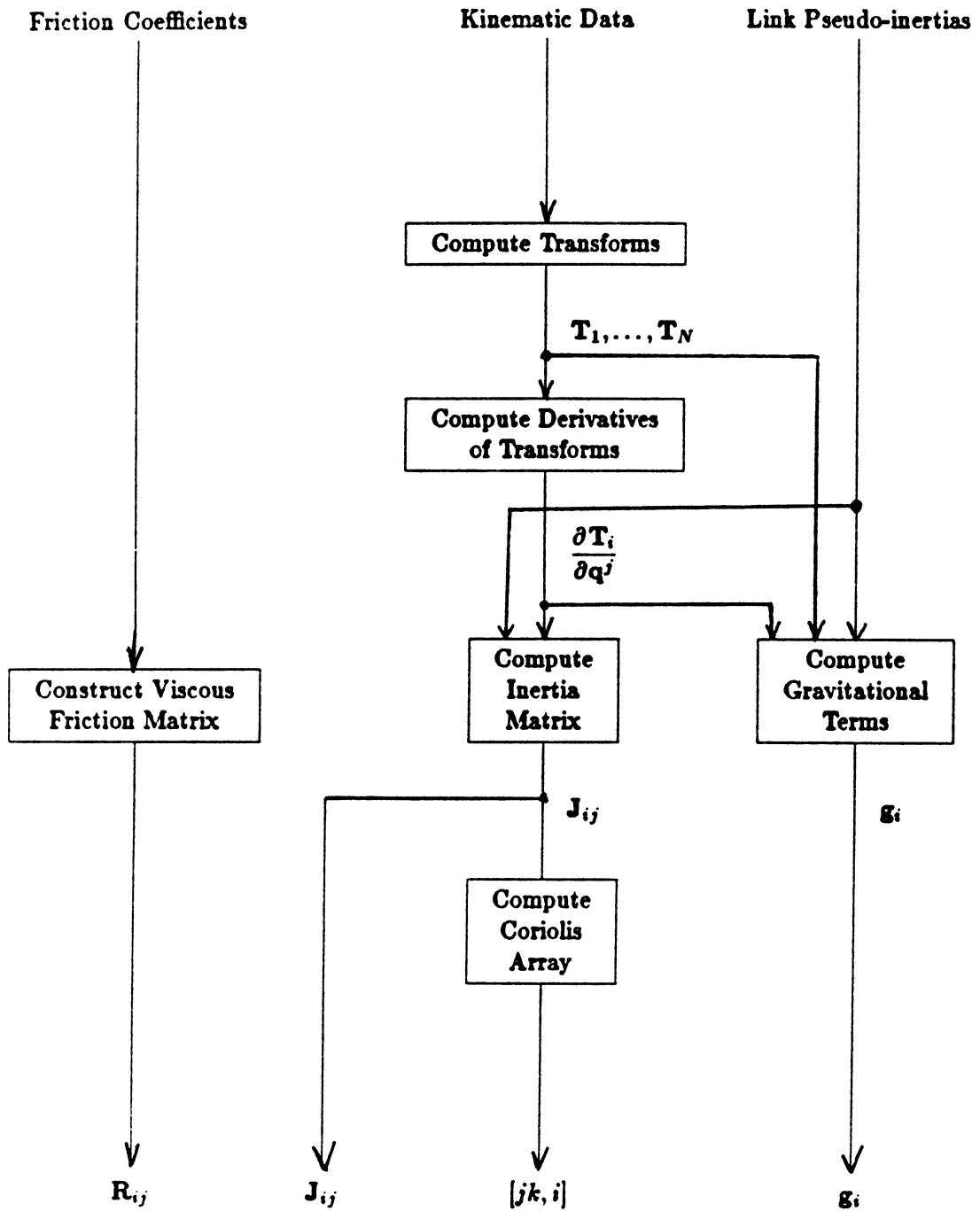


Figure 7.4.1. Schematic of Dynamic Equation Generator

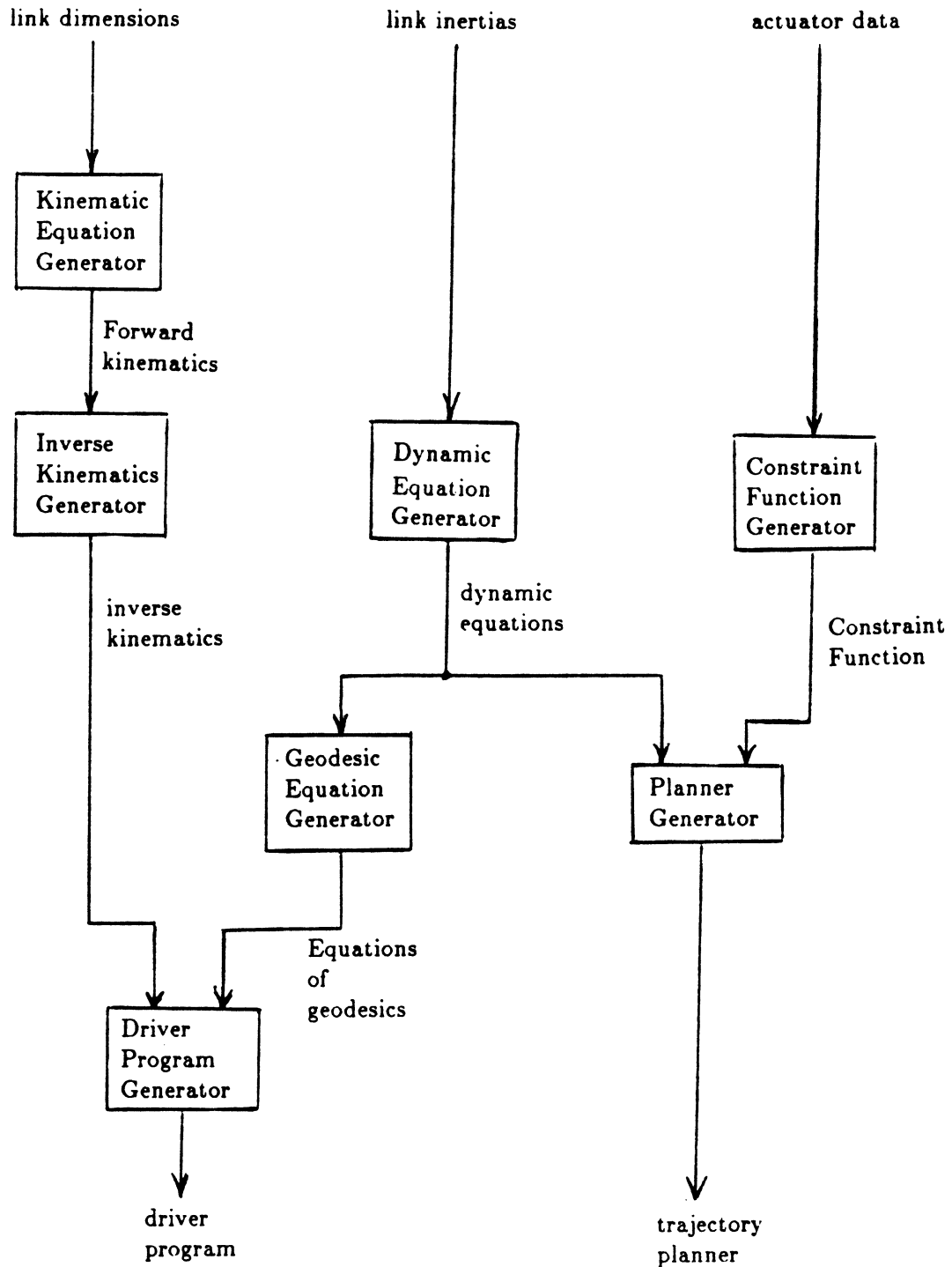


Figure 7.6.1. ATPG structure

## CHAPTER 8

### CONCLUSIONS

The aim of this thesis was to obtain methods for generating optimal point-to-point motion strategies for robot manipulators. Solving the optimal motion planning problem in its entirety will undoubtedly require much effort from many people, but the solutions of some significant subproblems have been presented here.

Most of the thesis has been devoted to the trajectory planning problem, or the problem of assigning timing information to a predefined geometric path. Three methods have been presented for the solution of the trajectory planning problem: the phase-plane method, the perturbation algorithm, and the dynamic programming method. The phase plane method generates timing information which is optimal in the minimum-time sense. It may be applied to most robots which are driven by D.C. torque motors, but cannot be used if there are significant constraints on the derivatives of the joint torques or if the joint torque limits interact with one another.

The perturbation method is also a minimum-time algorithm, though it could potentially be adapted to minimize other cost functions. The types of torque constraints that it can accommodate are much more general than those that can be used by the phase plane method, and the perturbation algorithm is structured in such a way that the torque constraints and the robot dynamics can be isolated as separate



program sections or modules, making implementation for a wide variety of robots very easy.

The dynamic programming method is very general; it can, in principle, optimize virtually any cost function under very general torque constraints. However, the problem must be discretized, and when the discretization is fine enough to give accurate results, the time required by the algorithm becomes prohibitive. On the other hand, a speed-up technique has been suggested which should considerably reduce the computation times. (Dynamic programming with this speed-up technique strongly resembles the perturbation algorithm.)

While solving the trajectory planning algorithm for a robot with known dynamics is an important step in optimal motion planning, the dynamic equations of a robot are usually known only approximately. Since minimum-time trajectory planners generate joint torques which saturate the actuators, it is possible, if the nominal and actual dynamic models vary too much, that the torques required to keep the robot in the right place at the right time will exceed the torques computed using the nominal dynamic model. Therefore possible differences between the actual and nominal dynamic models must be taken into account by the trajectory planner. This thesis has presented a method for doing this. By using a parameterized dynamic model, torque constraints can be checked for all dynamic models with parameter values which fall into some range, rather than for a dynamic model corresponding to a single set of parameter values. This allows the computation of the fastest trajectory which guarantees that the required torques are realizable for *all* dynamic models in the desired range.

One practical aspect of the trajectory planning problem, the automatic generation of trajectory planners, has been discussed. Writing a trajectory planner is a laborious task, and should be automated; otherwise, the effort required to write the trajectory planner may outweigh the savings realized by using optimal trajectories. A framework for writing a trajectory planner generator has been presented here, including suggestions on the choice of data structures and trajectory planning algorithms. The task of writing a trajectory planner generator will certainly be a major effort, but it poses no theoretical problems.

Finally, the problem of generating optimal geometric paths was investigated. The problem considered here was that of determining the geometric path which takes the robot from a given starting point to a given destination in the least time, given an unobstructed workspace. Though the problem was not solved completely, bounds on the traversal time were computed for a given geometric path, and these bounds could be minimized. The approximation obtained by minimizing these bounds seems to be fairly good, as shown by the simulation results. In addition, the sub-optimal paths produced by this method have an appealing intuitive interpretation: geometric paths with short traversal times are short in length and low in curvature. At the very least, this provides a good heuristic for finding low-cost geometric paths.

The conclusions of most theses state that much work remains to be done, and this thesis will uphold that tradition. The trajectory planning problem has been explored fairly thoroughly in this thesis, and the paradigm used here for creating trajectory planners (parameterize the desired path to reduce the dimensionality of the problem, then apply your favorite optimization technique) has proven to be quite

successful. On the other hand, a number of control and planning issues remain unresolved. Some physical phenomena, such as Coulomb friction, are difficult to model, and how the introduction of such phenomena affect the results presented here is still unknown. At the spatial planning level, much work remains to be done also. Exact methods for computing globally optimal geometric paths in the presence of obstacles have yet to be found; the process of obtaining non-optimal collision-free paths is still an open research topic, and finding paths which are guaranteed to be optimal may prove to be prohibitively costly. If this is the case, then perhaps the results derived here will be of some use as heuristics for finding near-optimal paths in an efficient way.

## **APPENDIX**

## APPENDIX

## Kinematics and Dynamics of the PACS Robot

The PACS (Programmable Assembly Cybernetics System) robot is a cylindrical robot arm, and was manufactured by the Bendix Corporation. (Figures A.1 and A.2 show the configuration of the first three joints of the PACS arm.) The dynamics of the first three joints of the PACS robot will be derived here, and the characteristics of the actuators for those joints will be given. The  $T_3$  matrix, which relates end-effector to world coordinates is obtained, and the partial derivatives end error coefficients required in Chapter 6 are calculated. All these derivations apply equally well to the 2-degree-of-freedom polar robot used in the numerical examples in Section 4.3; only numerical values need to be changed.

The simplest way to obtain the dynamics for the PACS arm is to obtain an expression for the kinetic and potential energies of the arm by *ad hoc* methods. To find the kinetic energy of the arm, first note that there is no coupling between the  $z$  joint and the  $r$  and  $\theta$  joints. We may therefore treat the  $z$  joint separately. The  $r$  and  $\theta$  joints are essentially the same as those of the two DOF polar robot. This robot consists of a rotating fixture with moment of inertia  $J_\theta$  through which slides a uniformly dense rod of length  $L_r$  and mass  $M_r$ . On the end of the rod is a payload with mass  $M_p$  and moment of inertia  $J_p$ . The payload is symmetric, and has length  $L_p$ . The moment of inertia of the sliding rod about the axis of rotation of the  $\theta$  joint is

$$J_r = M_r \left[ r^2 - (L_r + 2L_p)r + \left\{ L_p^2 + L_r L_p + \frac{L_r^2}{3} \right\} \right]$$

Adding up the kinetic energies of the various parts of the robot,

$$KE = \frac{1}{2} \left[ (J_\theta + J_r + J_p) \dot{\theta}^2 + M_r \dot{r}^2 + M_p \left\{ \dot{r}^2 + r^2 \dot{\theta}^2 \right\} \right]$$

The additional kinetic energy due to  $z$  axis motion is just  $\frac{1}{2} (M_p + M_r + M_f) \dot{z}^2$ . If

we define the constants

$$J_t = J_\theta + J_p + M_r \left\{ L_p^2 + L_r L_p + \frac{r^2}{3} \right\}$$

$$M_t = M_r + M_p$$

$$K = M_r (L_r + 2L_p)$$

$$M_z = M_r + M_p + M_f$$

then the inertia matrix is

$$\mathbf{J}_{ij} = \begin{bmatrix} J_t - Kr + M_t r^2 & 0 & 0 \\ 0 & M_t & 0 \\ 0 & 0 & M_z \end{bmatrix} \quad (\text{A.1})$$

where  $\mathbf{q}^1 = \theta$ ,  $\mathbf{q}^2 = r$ , and  $\mathbf{q}^3 = z$ . The Christoffel symbols of the first kind (Coriolis coefficients) are found by differentiating  $\mathbf{J}_{ij}$ . The non-zero coefficients are

$$[12,1] = [21,1] = M_t r - \frac{K}{2} \quad (\text{A.2})$$

$$[11,2] = \frac{K}{2} - M_t r. \quad (\text{A.3})$$

The gravitational forces  $\mathbf{g}_i$  are easily seen to be zero for the  $r$  and  $\theta$  joints, and  $M_z g$  for the  $z$  joint, where  $g$  is the acceleration due to gravity.

The PACS actuators will be modelled as ideal motors. The circuit for each joint drive is as shown in Figure A.3. It consists of a voltage source, a resistance  $R^m$ , an inductance  $L$ , and an ideal motor, i.e., a device which generates a torque proportional to the current passing through it. The voltage source is the power supply, the resistance is the sum of the voltage source resistance and the motor winding resistance, and the inductance is the inductance of the motor windings.

It will be assumed here that the inductance  $L$  can be neglected. This frequently is the case for D.C. motors, since the electrical time constant of such systems is generally much shorter than the mechanical time constant. Given that the torque  $\tau$  is proportional to the current, i.e.,  $\tau = k_m I$ , it can be shown from conservation of power that the voltage  $V_m$  across the ideal motor is just  $k_m \omega$ , where  $\omega$  is angular velocity. Since, if the motor is not in saturation,  $\tau = k_m I$  and

$$I = \frac{V_s - V_m}{R^m} = \frac{V_s - k_m \omega}{R^m} \quad \text{where } V_s \text{ is the source voltage, we can solve for}$$

torque in terms of voltage and angular velocity, giving

$$\tau = \frac{k_m}{R^m} V_s - \frac{k_m^2}{R^m} \omega. \quad (\text{A.4})$$

Assuming the power supply has constant voltage limits of  $V^{\min}$  and  $V^{\max}$ , this gives torque limits of

$$\frac{k^m}{R^m} V^{\min} - \frac{(k^m)^2}{R^m} \omega \leq \tau \leq \frac{k^m}{R^m} V^{\max} - \frac{(k^m)^2}{R^m} \omega. \quad (\text{A.5})$$

In addition, at some point the iron in the motor saturates, with the result that increasing the current through the motor has no effect on the torque. This yields two more (constant) torque limits, so we also require that

$$-\tau^{\text{sat}} \leq \tau \leq \tau^{\text{sat}}. \quad (\text{A.6})$$

Taking the gear ratio  $k^g$  into account, this gives torque limits of

$$u_i^{\min} = \max \left( -\frac{\tau_i^{\text{sat}}}{k_i^g}, \frac{k_i^m}{R_i^m k_i^g} V_i^{\min} - \frac{(k_i^m)^2}{R_i^m (k_i^g)^2} \frac{d q^i}{d \lambda} \mu \right) \quad (\text{A.7})$$

and

$$u_i^{\max} = \min \left( \frac{\tau_i^{\text{sat}}}{k_i^g}, \frac{k_i^m}{R_i^m k_i^g} V_i^{\max} - \frac{(k_i^m)^2}{R_i^m (k_i^g)^2} \frac{d q^i}{d \lambda} \mu \right) \quad (\text{A.8})$$

Table A.1 gives the dynamic and actuator characteristics for the polar manipulator, while Table A.2 gives the data for the for the PACS arm.

The kinematics of the PACS arm, required in Chapter 6, are quite simple. If the coordinate frames of the base and hand are as shown in Figure A.2, then the coordinate transform  $\mathbf{T}_3$  is easily shown to be

$$\mathbf{T}_3 = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & -r \sin \theta \\ \sin \theta & 0 & \cos \theta & r \cos \theta \\ 0 & -1 & 0 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{A.9})$$

The partial derivatives of  $\mathbf{T}_3$  are



$$\frac{\partial \mathbf{T}_3}{\partial z} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial \mathbf{T}_3}{\partial \theta} = \begin{bmatrix} -\sin \theta & 0 & -\cos \theta & -r \cos \theta \\ \cos \theta & 0 & -\sin \theta & -r \sin \theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (\text{A.10})$$

$$\frac{\partial \mathbf{T}_3}{\partial r} = \begin{bmatrix} 0 & 0 & 0 & -\sin \theta \\ 0 & 0 & 0 & \cos \theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \quad (\text{A.11})$$

There are two non-zero second partials; they are

$$\frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} = \begin{bmatrix} -\cos \theta & 0 & \sin \theta & r \sin \theta \\ -\sin \theta & 0 & -\cos \theta & -r \cos \theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (\text{A.12})$$

and

$$\frac{\partial^2 \mathbf{T}_3}{\partial r \partial \theta} = \begin{bmatrix} 0 & 0 & 0 & -\cos \theta \\ 0 & 0 & 0 & -\sin \theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

We are now in a position to compute the  $\delta M_i$ , as described in Chapter 6. If we let

$\mathbf{H} = \Delta I_N$ , the pseudo-inertia error, and define

$$\mathbf{m}_{ij} = \text{Tr} \left( \frac{\partial \mathbf{T}_N}{\partial \mathbf{q}^j} \mathbf{H} \frac{\partial \mathbf{T}_N^T}{\partial \mathbf{q}^i} \right)$$

then we have

$$\delta M_i = \mathbf{m}_{ij} \frac{d \mathbf{q}^j}{d \lambda}.$$

Computing the  $m_{ij}$ ,

$$m_{zz} = \text{Tr} \left( \frac{\partial \mathbf{T}_3}{\partial z} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial z} \right) \quad \text{and} \quad \frac{\partial \mathbf{T}_3}{\partial z} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial z} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & H_{44} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Taking the trace of this gives  $m_{zz} = H_{44}$ .

$$m_{z\theta} = m_{\theta z} = \text{Tr} \left( \frac{\partial \mathbf{T}_3}{\partial z} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial \theta} \right)$$

$$\frac{\partial \mathbf{T}_3}{\partial z} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial \theta} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ H_{14} & H_{24} & H_{34} & H_{44} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\cos\theta & -\sin\theta & 0 & 0 \\ -r \cos\theta & -r \sin\theta & 0 & 0 \end{bmatrix}.$$

The trace of this matrix is zero.

$$m_{zr} = m_{rz} = \text{Tr} \left( \frac{\partial \mathbf{T}_3}{\partial z} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial r} \right)$$

$$\frac{\partial \mathbf{T}_3}{\partial z} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial r} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ H_{14} & H_{24} & H_{34} & H_{44} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \end{bmatrix}$$

The trace of this matrix also is zero.

$$m_{\theta\theta} = \text{Tr} \left( \frac{\partial \mathbf{T}_3}{\partial \theta} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial \theta} \right)$$

$$\frac{\partial \mathbf{T}_3}{\partial \theta} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial \theta} = \begin{bmatrix} -\sin\theta & 0 & -\cos\theta & -r \cos\theta \\ \cos\theta & 0 & -\sin\theta & -r \sin\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{12} & H_{22} & H_{23} & H_{24} \\ H_{13} & H_{23} & H_{33} & H_{34} \\ H_{14} & H_{24} & H_{34} & H_{44} \end{bmatrix}$$

The diagonal entries of this matrix are

$$\begin{aligned} e_{11} = & H_{11} \sin^2\theta + 2H_{13} \sin\theta \cos\theta + 2H_{14} r \sin\theta \cos\theta \\ & + H_{33} \cos^2\theta + 2H_{34} r \cos^2\theta + H_{44} r^2 \cos^2\theta \end{aligned}$$

$$\begin{aligned} e_{22} = & H_{11} \cos^2\theta - 2H_{13} \sin\theta \cos\theta - 2H_{14} r \sin\theta \cos\theta \\ & + H_{33} \sin^2\theta + 2H_{34} r \sin^2\theta + H_{44} r^2 \sin^2\theta \end{aligned}$$

$$e_{33} = e_{44} = 0.$$

Adding these up to get the trace,

$$m_{\theta\theta} = \text{Tr} \left( \frac{\partial \mathbf{T}_3}{\partial \theta} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial \theta} \right) = H_{11} + H_{33} + 2rH_{34} + r^2H_{44}$$

$$m_{\theta r} = m_{r\theta} = \text{Tr} \left( \frac{\partial \mathbf{T}_3}{\partial \theta} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial r} \right).$$

The diagonal entries of  $\frac{\partial \mathbf{T}_3}{\partial \theta} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial r}$  are

$$e_{11} = H_{14}\sin^2\theta + H_{34}\sin\theta\cos\theta + H_{44}r\sin\theta\cos\theta$$

$$e_{22} = H_{14}\cos^2\theta - H_{34}\sin\theta\cos\theta - H_{44}r\sin\theta\cos\theta$$

$$e_{33} = e_{44} = 0.$$

Adding these up gives  $m_{\theta r} = H_{14}$ .

$$m_{rr} = Tr \left( \frac{\partial \mathbf{T}_3}{\partial r} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial r} \right)$$

$$\frac{\partial \mathbf{T}_3}{\partial r} \mathbf{H} \frac{\partial \mathbf{T}_3^T}{\partial r} = \begin{bmatrix} 0 & 0 & 0 & -\sin\theta \\ 0 & 0 & 0 & \cos\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{12} & H_{22} & H_{23} & H_{24} \\ H_{13} & H_{23} & H_{33} & H_{34} \\ H_{14} & H_{24} & H_{34} & H_{44} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} H_{44}\sin^2\theta & -H_{44}\sin\theta\cos\theta & 0 & 0 \\ -H_{44}\sin\theta\cos\theta & H_{44}\cos^2\theta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The trace of this is  $m_{rr} = H_{44}$ .

We can now find the  $\delta M_i$ . We have

$$\delta M_z = H_{44} \frac{dz}{d\lambda}$$

$$\delta M_\theta = (H_{11} + H_{33} + 2rH_{34} + r^2H_{44}) \frac{d\theta}{d\lambda} + H_{14} \frac{dr}{d\lambda}$$

$$\delta M_r = H_{14} \frac{d\theta}{d\lambda} + H_{44} \frac{dr}{d\lambda}$$

If we define  $c_{ijk}$  by  $c_{ijk} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_N}{\partial q^j \partial q^k} \mathbf{H} \frac{\partial \mathbf{T}_N^T}{\partial q^i} \right)$ , then we have

$$\delta Q_i = m_{ij} \frac{d^2 f^j}{d\lambda^2} + c_{ijk} \frac{df^j}{d\lambda} \frac{df^k}{d\lambda}.$$

We now compute the  $c_{ijk}$ . Only six cases need to be considered, since all but two of the second partials of  $\mathbf{T}_3$  are zero.

$$c_{z\theta\theta} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial z} \right)$$

$$\frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial z} = \begin{bmatrix} -\cos\theta & 0 & \sin\theta & r \sin\theta \\ -\sin\theta & 0 & -\cos\theta & -r \cos\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{12} & H_{22} & H_{23} & H_{24} \\ H_{13} & H_{23} & H_{33} & H_{34} \\ H_{14} & H_{24} & H_{34} & H_{44} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The trace of this matrix is zero.

$$c_{\theta\theta\theta} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial \theta} \right).$$

The diagonal terms of  $\frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial \theta}$  are

$$e_{11} = H_{11} \sin\theta \cos\theta - H_{13} \sin^2\theta - H_{14} r \sin^2\theta + H_{13} \cos^2\theta - H_{33} \sin\theta \cos\theta$$

$$- H_{34} r \sin\theta \cos\theta + H_{14} r \cos^2\theta - H_{34} r \sin\theta \cos\theta - H_{44} r^2 \sin\theta \cos\theta$$

$$e_{22} = -H_{11} \sin\theta \cos\theta - H_{13} \cos^2\theta - H_{14} r \cos^2\theta + H_{13} \sin^2\theta + H_{33} \sin\theta \cos\theta$$

$$+ H_{34} r \sin\theta \cos\theta + H_{14} r \sin^2\theta + H_{34} r \sin\theta \cos\theta + H_{44} r^2 \sin\theta \cos\theta$$

$$c_{33} = c_{44} = 0$$

The trace is zero.

$$c_{r\theta\theta} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial r} \right).$$

The diagonal terms of  $\frac{\partial^2 \mathbf{T}_3}{\partial \theta^2} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial r}$  are

$$c_{11} = H_{14} \sin \theta \cos \theta - H_{34} \sin^2 \theta - H_{44} r \sin^2 \theta$$

$$c_{22} = -H_{14} \sin \theta \cos \theta - H_{34} \cos^2 \theta - H_{44} r \cos^2 \theta$$

$$c_{33} = c_{44} = 0.$$

The trace therefore is  $c_{r\theta\theta} = -H_{34} - H_{44} r$ .

$$c_{z\theta r} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_3}{\partial \theta \partial r} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial z} \right)$$

$$\frac{\partial^2 \mathbf{T}_3}{\partial \theta \partial r} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial z} = \begin{bmatrix} 0 & 0 & 0 & -\cos \theta \\ 0 & 0 & 0 & -\sin \theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{12} & H_{22} & H_{23} & H_{24} \\ H_{13} & H_{23} & H_{33} & H_{34} \\ H_{14} & H_{24} & H_{34} & H_{44} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The trace of this matrix is zero.

$$c_{\theta\theta r} = \text{Tr} \left( \frac{\partial^2 \mathbf{T}_3}{\partial \theta \partial r} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial \theta} \right).$$

The diagonal terms of  $\frac{\partial^2 \mathbf{T}_3}{\partial \theta \partial r} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial \theta}$  are

$$e_{11} = H_{14} \sin \theta \cos \theta + H_{34} \cos^2 \theta + H_{44} r \cos^2 \theta$$

$$e_{22} = -H_{14} \sin \theta \cos \theta + H_{34} \sin^2 \theta + H_{44} r \sin^2 \theta$$

$$e_{33} = e_{44} = 0.$$

The trace is  $c_{\theta\theta r} = H_{34} + H_{44} r$ .

$$c_{r\theta r} = T_r \left( \frac{\partial^2 \mathbf{T}_3}{\partial \theta \partial r} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial r} \right).$$

The diagonal terms of  $\frac{\partial^2 \mathbf{T}_3}{\partial \theta \partial r} \mathbf{H} \frac{\partial \mathbf{T}_3}{\partial r}$  are

$$e_{11} = H_{44} \sin \theta \cos \theta$$

$$e_{22} = -H_{44} \sin \theta \cos \theta$$

$$e_{33} = e_{44} = 0.$$

The trace is zero.

Calculating the  $\delta Q_i$ ,

$$\delta Q_z = H_{44} \frac{d^2 z}{d\lambda^2}$$

$$\delta Q_\theta = (H_{11} + H_{33} + 2rH_{34} + r^2H_{44}) \frac{d^2 \theta}{d\lambda^2} + H_{14} \frac{d^2 r}{d\lambda^2} + 2(H_{34} + rH_{44}) \frac{dr}{d\lambda} \frac{d\theta}{d\lambda}$$

$$\delta Q_r = H_{14} \frac{d^2 \theta}{d\lambda^2} + H_{44} \frac{d^2 r}{d\lambda^2} - (H_{34} + rH_{44}) \left( \frac{d\theta}{d\lambda} \right)^2.$$

The  $\delta S_i$ , the gravitational error coefficients, are easily seen to be  $\delta S_z = H_{44}g$  and  $\delta S_\theta = \delta S_r = 0$ , where  $g$  is the acceleration due to gravity.



Constant	Description	Value
$J_\theta$	Moment of inertia of $\theta$ joint	$10^{-3} \text{ Kg-M}$
$M_r$	Mass of sliding rod	4.0 Kg.
$L_r$	Length of rod	2.0 M.
$J_p$	Moment of inertia of payload	$10^{-5} \text{ Kg. -M}$
$M_p$	Payload mass	1.0 Kg.
$L_p$	Length of payload	0.1
$u_r^{\max}$	Maximum force on $r$ joint	$1.0 \text{ Kg. -M/sec}^2$
$u_\theta^{\max}$	Maximum torque on $\theta$ joint	$1.0 \text{ Kg. -M}^2/\text{sec}^2$
$k_r$	Friction coefficient of $r$ joint	0.0 (low friction)
	Friction coefficient of $r$ joint	15.0 (high friction)
$k_\theta$	Friction coefficient of $\theta$ joint	0.0

Table A.1. Characteristics of the polar manipulator

Parameter	Description	Value
$\tau_{\theta}^{sat}$	Saturation torque of $\theta$ motor	2.0 Nt.-M.
$\tau_r^{sat}$	Saturation torque of $r$ motor	0.05 Nt.-M.
$\tau_z^{sat}$	Saturation torque of $z$ motor	2.0 Nt.-M.
$V_{\theta}^{min}$	Lower voltage limit for $\theta$ joint	-40 v.
$V_r^{min}$	Lower voltage limit for $r$ joint	-40 v.
$V_z^{min}$	Lower voltage limit for $z$ joint	-40 v.
$V_{\theta}^{max}$	Upper voltage limit for $\theta$ joint	40 v.
$V_r^{max}$	Upper voltage limit for $r$ joint	40 v.
$V_z^{max}$	Upper voltage limit for $z$ joint	40 v.
$k_{\theta}^g$	Gear ratio for $\theta$ drive	0.01176
$k_r^g$	Gear ratio for $r$ drive	0.00318 Meters/radian
$k_z^g$	Gear ratio for $z$ drive	0.00318 Meters/radian
$k_{\theta}^m$	Motor constant for $\theta$ joint	0.0397 Nt.-M./amp
$k_r^m$	Motor constant for $r$ joint	$0.79557 \times 10^{-3}$ Nt.-M./amp
$k_z^m$	Motor constant for $z$ joint	0.0397 Nt.-M./amp
$R_{\theta}^m$	Motor and power supply resistance, $\theta$ joint	1 $\Omega$
$R_r^m$	Motor and power supply resistance, $r$ joint	1 $\Omega$
$R_z^m$	Motor and power supply resistance, $z$ joint	1 $\Omega$
$k_{\theta}$	Friction coefficient of $\theta$ joint	8.0 Kg./sec.
$k_r$	Friction coefficient of $r$ joint	4.0 Kg./sec.
$k_z$	Friction coefficient of $z$ joint	1.0 Kg./sec.
$M_i$	Mass of $r$ joint	10.0 Kg.
$M_z$	Mass of $z$ joint	40.0 Kg.
$J_i$	Moment of inertia around $\theta$ axis	12.3183 Kg.-M. <sup>2</sup>
$K$	Moment of inertia offset term	3.0 Kg.-M.

Table A.2. Characteristics of the PACS arm

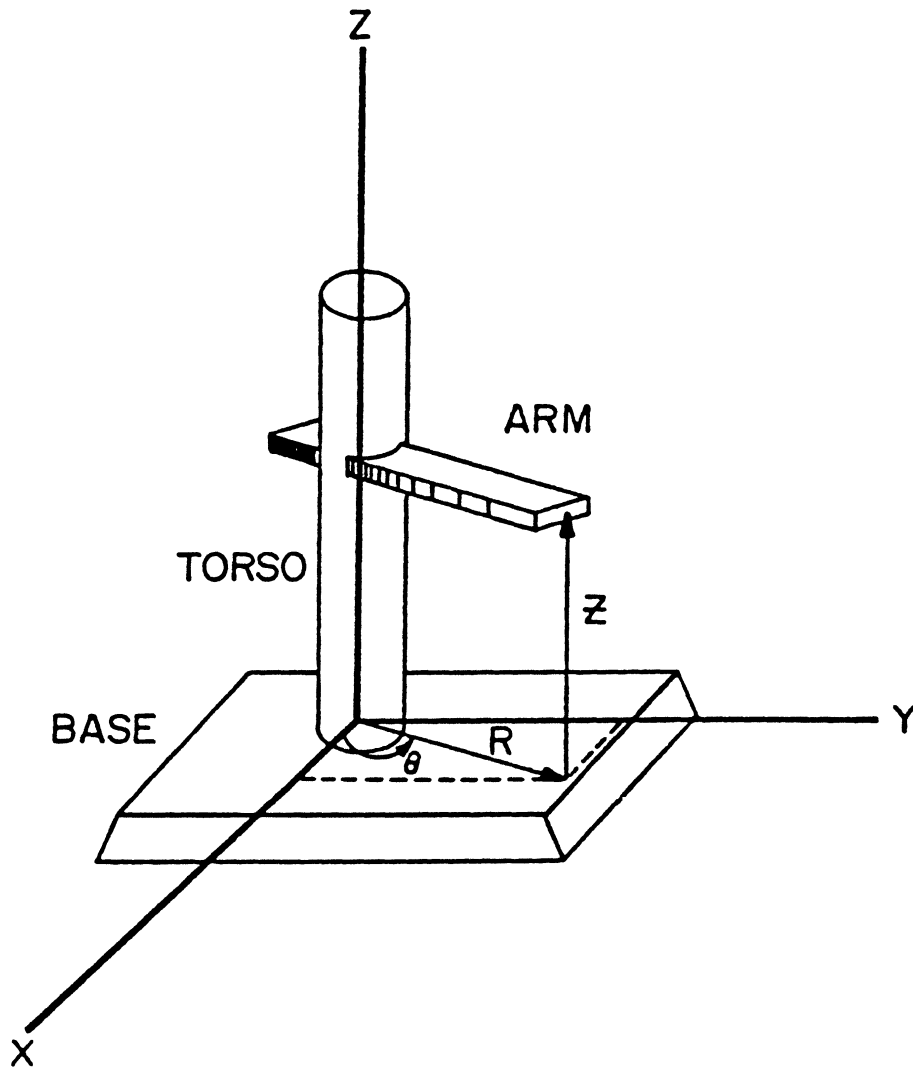


Figure A.1. Schematic of the first three links of the PACS robot

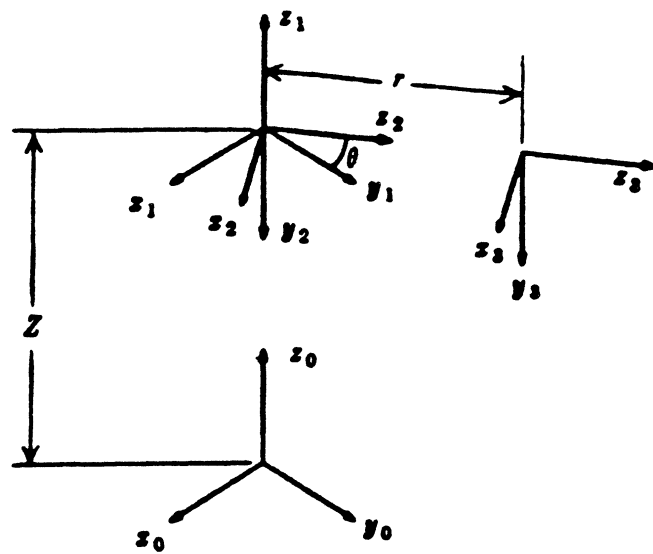


Figure A.2. Link coordinate frames of the PACS robot

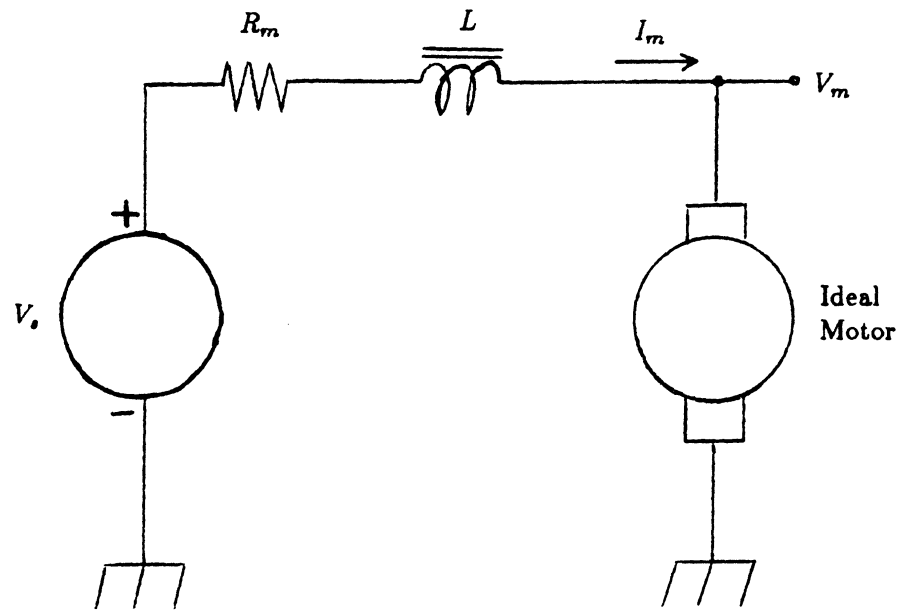


Figure A.3. PACS actuator circuit diagram

## **REFERENCES**

## REFERENCES

- [1] A. K. Bejczy and R. P. C. Paul, "Simplified Robot Arm Dynamics for Control," *Proceedings of the 20<sup>th</sup> Conference on Decision and Control*, pp. 261-262, Dec. 1981.
- [2] R. Bellman, "Functional Equations in the Theory of Dynamic Programming - VI, A Direct Convergence Proof," *Annals of Mathematics*, vol. 65, no. 2, pp. 215-223, March 1957.
- [3] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "On the Optimal Control of Robotic Manipulators with Actuator Constraints," *Proceedings of the 1983 Automatic Control Conference*, pp. 782-787, June 1983.
- [4] M. J. Chung and C. S. G. Lee, "An Adaptive Control Strategy for Computer-based Manipulators," report #RDS-TR-10-82, University of Michigan Center for Robotics and Integrated Manufacturing, Ann Arbor, MI, August 1982.
- [5] S. J. Derby, "Kinematic Elasto-Dynamic Analysis and Computer Graphic Simulation of General Purpose Robot Manipulators," Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, New York, August 1981.
- [6] S. Dubowsky, "On the Adaptive Control of Robotic Manipulators: The Discrete-Time Case," *Proceedings of the Joint Automatic Control Conference*, June 1981, section TA-2B.
- [7] S. Dubowsky and D. T. DesForges, "The Application of Model-Referenced Adaptive Control to Robotic Manipulators," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 101, pp. 193-200, September 1979.
- [8] E. G. Gilbert and D. W. Johnson, "Distance Functions and their Application to Robot Path Planning in the Presence of Obstacles," CRIM technical memo #RSD-TR-7-84, University of Michigan Center for Robotics and Integrated Manufacturing, Ann Arbor, MI., July 1984.
- [9] D. Ter Haar, *Elements of Hamiltonian Mechanics, Second edition*. Pergamon Press, 1971.

- [10] A. C. Hearn, "REDUCE User's Manual," Report UCP-19, University of Utah, 1973.
- [11] M. E. Kahn and B. Roth, "The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains," *ASME Journal of Dynamic Systems, Measurement, and Control*, pp. 164-172, September 1971.
- [12] B. K. Kim and K. G. Shin, "An Efficient Minimum-Time Robot Path Planning Under Realistic Constraints," *Proc. 1984 American Control Conf.*, pp. 296-303, June 1984.
- [13] B. K. Kim and K. G. Shin, "Minimum-time path planning for robot arms and their dynamics," *IEEE Trans. System, Man, and Cybernetics*, vol. SMC-15, no. 2, pp. 213-223, March/April 1985.
- [14] B. K. Kim and K. G. Shin, "An adaptive model following control of industrial manipulators," *IEEE Trans. Aerospace and Electronic Systems.*, vol. AES-19, no. 6, pp. 805-814, November 1983.
- [15] D. E. Kirk, *Optimal control theory: an introduction*. Englewood Cliffs, New Jersey: Prentice-Hall, 1971.
- [16] A. J. Koivo and T. H. Guo, "Control of a Robotic Manipulator with Adaptive Controller," *Proceedings of the 20<sup>th</sup> IEEE Conference on Decision and Control*, pp. 271-276, Dec. 1981.
- [17] A. J. Koivo and T. H. Guo, "Adaptive Linear Controller for Robotic Manipulators," *IEEE Transactions on Automatic Control*, vol. AC-28, no. 2, pp. 162-170, February 1983.
- [18] V. Lakshmikantham and S. Leela, *Differential and Integral Inequalities*. New York: Academic Press, 1969.
- [19] C.-S. Lin, P.-R. Chang, and J. Y. S. Luh, "Formulation and optimization of cubic polynomial joint trajectories for mechanical manipulators," *Proc. 21 CDC*, pp. 330-335, Dec. 1982.
- [20] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," A. I. memo 605, MIT Artificial Intelligence Laboratory, December 1980.
- [21] T. Lozano-Perez, "Automatic Planning of Manipulator Transfer Movements," A. I. memo 606, MIT Artificial Intelligence Laboratory, December 1980.



- [22] J. Y. S. Luh and C. S. Lin, "Optimum Path Planning for Mechanical Manipulators," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 102, pp. 142-151, June 1981.
- [23] J. Y. S. Luh and M. W. Walker, "Minimum-Time Along The Path for a Mechanical Arm," *Proceedings of the IEEE Conference on Decision and Control*, pp. 755-759, December 1977.
- [24] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "Resolved-Acceleration Control of Mechanical Manipulators," *IEEE Transactions on Automatic Control*, vol. AC-25 No. 3, pp. 468-474, June 1980.
- [25] J. Y. S. Luh and C. E. Campbell, "Collision-Free Path Planning for Industrial Robots," *Proceedings of the 21<sup>st</sup> CDC*, pp. 84-88, Dec. 8-10, 1982.
- [26] J. Y. S. Luh and C. S. Lin, "Automatic Generation of Dynamic Equations for Mechanical Manipulators," *Proceedings of the Joint Automatic Control Conference*, June 1981, Section TA-2D.
- [27] J. Moses, "Algebraic Simplification: A Guide for the Perplexed," *Proceedings of the 2<sup>nd</sup> ACM Symposium on Symbolic and Algebraic Manipulation*, pp. 282-304, 1971.
- [28] R. P. C. Paul, *Robot manipulators: Mathematics, programming, and control*. Cambridge, Mass.: MIT Press, 1981.
- [29] K. G. Shin and N. D. McKay, "Minimum-Time Control of a Robotic Manipulator with Geometric Path Constraints," *Proceedings of the 22<sup>nd</sup> CDC*, pp. 1449-1457, Dec. 1983.
- [30] K. G. Shin and N. D. McKay, "Minimum-Time Control of a Robotic Manipulator with Geometric Path Constraints," *IEEE Transactions on Automatic Control*, vol. AC-30, no. 6, pp. 531-541, June 1985.
- [31] W. M. Silver, "On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators," *Proceedings of the Joint Automatic Control Conference*, June 1981, section TA-2A.
- [32] J. L. Synge and A. Schild, *Tensor Calculus*. New York: Dover Publications, 1978.



- [33] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," *IEEE Transactions on Man-Machine Systems*, vol. MMS-10, no. 2, pp. 47-53, June 1969.