A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators

KANG G. SHIN, SENIOR MEMBER, IEEE, AND NEIL D. MCKAY

***Abstract*—This paper presents a solution to the problem of minimizing the cost of moving a robotic manipulator along a specified geometric path subject to input torque/force constraints, taking the coupled, nonlinear dynamics of the manipulator into account. The proposed method uses dynamic programming (DP) to find the positions, velocities, accelerations, and torques that minimize cost. Since the use of parametric functions *reduces* the dimension of the state space from $2n$ for an n-jointed manipulator, to *two*, the DP method does not suffer from the "curse of dimensionality." While maintaining the elegance of our previous trajectory planning method, we have developed the DP method for the general case where 1) the actuator torque limits are dependent on one another, 2) the cost functions can have an arbitrary form, and 3) there are constraints on the *jerk*, or derivative of the acceleration. Also, we have shown that the DP solution converges as the grid size decreases.**

As numerical examples, the trajectory planning method is simulated for the first three joints of the PACS arm, which is a cylindrical arm manufactured by the Bendix Corporation.

I. Introduction

EFFICIENT control of industrial robots is a key to the success of contemporary industrial automation which is built around robots, more specifically robotic manipulators. The problem of robot control is very complex because of the nonlinearity and couplings in robot dynamics and is therefore usually solved by a *two-stage optimization*. The first stage is called *path* or *trajectory planning*, and the second stage is called *control* or *trajectory* (or *path*) *tracking*. The trajectory tracker is responsible for making the robot's actual position and velocity match desired values of position and velocity [9], [16]; the desired values are provided to the tracker by the trajectory planner. The trajectory planner receives as input spatial path descriptor [5], [6] from which it calculates a time history of the desired positions, velocities, and joint torques. Note that the reason for dividing the control scheme in this way is for tractability of the robot control problem.

Earlier trajectory planners, such as those presented in [7], [8] use linear and/or nonlinear programming to generate desired positions, velocities, and accelerations. These methods assume that the desired path is given in terms of the path's endpoints and a set of intermediate points or corner points. Along each segment of the path the (constant) maximum accelerations and velocities are given. In general, these constant bounds may be quite inaccurate for some parts of the segment, since worst-case bounds for the whole segment must be used. Additionally, these methods provide no rigorous means of obtaining the maximum accelerations and velocities; thus, once the trajectory planning process has been completed, the solution must be validated to make certain that the robot's capabilities are not exceeded.

The algorithms presented in [2] and [12], [13] do not suffer from either of these problems, since they calculate acceleration and velocity limits directly from the given path and the robot's dynamic equations and actuator characteristics. The method presented in [2] uses phase plane plots to construct the optimal (in the minimum-time sense) trajectory for a given robot path. It is assumed that the path is given in parameterized form, and that the actuator torque limits are functions only of the position and velocity of the manipulator. This method will solve the minimum-time trajectory planning problem for a wide variety of robots, provided the actuator torque limits are independent of one another. We have developed a similar method in [12] independently of [2]. However, the technique for determining switching points in [12] is different from that used in [2]. It is more direct, but requires that the torque bounds be at most quadratic in the velocity. In practice, this is usually the case, so the limitations on the forms of the torque bounds should not significantly limit the applicability of the method. Moreover, the method in [12] can handle the general case where the feasible regions in the phase plane are not simply connected, whereas that in [2] cannot.

While these methods are quite elegant, they do have some drawbacks. First, they work only for minimum time problems. 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. Second, it is assumed that the joint torques can be changed instantaneously. This is only approximately true, and indeed it is desirable to limit the derivatives of the joint torques (or, equivalently, the jerk, or derivative of the acceleration) to prevent excessive mechanism wear. Third, they are unable to handle the general case where the actuator torque limits are dependent on one another. This dependency occurs, for example, when a robot uses a common power supply for the servoamplifiers for all joints.

The correction of these deficiencies is the aim of this paper. The method proposed here is to use dynamic programming [3], rather than the methods described above, to find the optimal phase plane trajectory. Unlike those methods, dynamic programming places few restrictions on the cost function that is to be minimized. Putting limits on jerk is also possible, and interdependence of torque bounds can be handled fairly painlessly, as will be seen later. Since the use of parametric functions *reduces* the dimension of the state space from $2n$ for an n-jointed manipulator to *two*, the dynamic programming method does not suffer from the "curse of dimensionality."

The remainder of this paper is divided into four sections. Section II gives a detailed description of the problem. Section III presents a solution algorithm using dynamic programming. Section IV deals with the convergence of the dynamic programming algorithm. Section V presents numerical examples using the Bendix PACS arm. Using these numerical examples, first, we compare the results of the direct minimum-time phase plane methods [13] to those of the dynamic programming technique.

Manuscript received January 16, 1985; revised November 22, 1985. This paper is based on a prior submission of March 23, 1984. Paper recommended by Associate Editor, W. J. Book. This work was supported in part by the National Science Foundation under Grant ECS-8409938 and the U. S. Air Force Office of Scientific Research under Contracts F49620-82-C-0089 and F33615-85-C-5105.

K. G. Shin is with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

N. D. McKay was with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109. He is now with the Computer Science Department, General Motors Research Laboratories, Warren, MI 48090.

IEEE Log Number 8607827.

0018-9286/86/0600-0491$01.00 © 1986 IEEE

Second, we examine the sensitivity of the trajectory planning solution to the grid size. Third, the dynamic programming method is applied to the general case where 1) cost functions other than minimum time, and 2) coupling among the actuator torque bounds are considered. Finally, Section VI states conclusions drawn from our results.

II. Problem Statement

The goal of automation is to produce goods at as low a cost as possible. The minimum-cost trajectory planning (MCTP) in the two staged realization of manipulator control, i.e., planning first and tracking next, is thus of utmost importance to any effort in accomplishing the goal.

A loose statement of the MCTP problem is as follows:

> What control signals will drive a given robot from a given initial configuration to a given final configuration with as low a cost as possible, given constraints on the magnitudes and derivatives 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 [5], [6] is not a control theory problem in the normal sense, the problem of moving a mechanical system in 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.[1] 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 signals and their derivatives?

This form of the problem reduces the complexity of the control problem by introducing a single parameter that 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 a two-dimensional minimum-cost control problem with some state and input constraints.

Usually, a geometric path can be constructed by connecting endpoints and intermediate points with some means such as straight line segments or cubic splines. In such a case, the geometric path can be given in the form of a parameterized curve[2]

$$q^i = f^i(\lambda), \qquad 0 \le \lambda \le \lambda_{\max} \tag{2.1}$$

where q^i is the position of the ith joint, $q = [q^1, q^2, \cdots, q^n]^T$ represents the joint position vector for an n-jointed manipulator, and the initial and final points on the geometric path correspond to the points $\lambda = 0$ and $\lambda = \lambda_{\max}$. Also assume 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

$$u = (u_1, u_2, \cdots, u_n)^T \in E(q, \dot{q}) \tag{2.2}$$

where $\dot{q}$ is the first derivative of q with respect to time, and u_i is the ith 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 q and velocity $\dot{q}$ if and only if the torque vector u is in the set $E(q, \dot{q})$.[3]

In practice, it is desirable to limit the derivatives of the joint torque (or, equivalently, derivative of the acceleration, or the jerk) to prevent excessive mechanism wear. This need introduces inequalities

$$|\dot{u}_i| \le K_i \tag{2.3}$$

where K_i is a constant.

The manipulator dynamic equations usually take the form

$$u_i = J_{ij}\ddot{q}^j + C_{ijk}\dot{q}^j\dot{q}^k + R_{ij}\dot{q}^j + G_i \tag{2.4}$$

where the Einstein summation convention is used, and

J_{ij} ≡ the inertia matrix,
C_{ijk} ≡ the array of centrifugal and Coriolis coefficients,
R_{ij} ≡ the viscous friction matrix, and
G_i ≡ the gravitational loading vector.

If the equations of the parameterized path are plugged into the dynamic equations, then they become (see [12] for a detailed derivation)

$$u_i = J_{ij}\frac{df^j}{d\lambda}\dot{\mu} + \left(J_{ij}\frac{d^2f^j}{d\lambda^2} + C_{ijk}\frac{df^j}{d\lambda}\frac{df^k}{d\lambda}\right)\mu^2 + R_{ij}\frac{df^j}{d\lambda}\mu + G_i. \tag{2.5}$$

Here, μ is the time-derivative of the parameter λ, i.e., $\mu \equiv \dot{\lambda}$.

The cost C will be assumed to take the form

$$C = \int_0^{\lambda_{\max}} L(\lambda, \theta, u_i)\, d\lambda. \tag{2.6}$$

The MCTP problem then becomes that of minimizing (2.6) subject to (2.5) and the inequalities (2.2) and (2.3).

III. MCTP Using Dynamic Programming

To see how dynamic programming can be applied to this problem, first note that by using the parameterized path (2.1), the dimensionality of the problem has been reduced; there will be only *two* state variables λ and μ, *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 ($\lambda - \mu$ plane) into a discrete grid. Then, the costs of going from one point on the grid to the next must be calculated. Note that since u_i is a function of λ and μ as shown in (2.5), (2.6) are given strictly in terms of λ and μ; 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 (2.1) and (2.5).

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, rewrite (2.5) in a more convenient form

$$u_i = M_i\dot{\mu} + Q_i\mu^2 + R_i\mu + S_i \tag{3.1}$$

where

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

$$R_i \equiv R_{ij}\frac{df^j}{d\lambda}, \text{ and } S_i \equiv G_i.$$

[1] The task planner generates a sequence of Cartesian points and additional intermediate knot points which, if necessary, are transformed *pointwise* to points in joint space [11]. These joint points are then interpolated to form a geometric path in joint spaces, e.g., [4]. However, this process is not in the scope of the present paper.

[2] Note, however, that the determination of an "optimal" (in some sense) parametric function f^i is an open research problem. See [14] for an example.

[3] This dependence can be seen easily in the case of DC servodriven manipulators.

Note that M_i, Q_i, R_i, and S_i are all functions of the parameter λ, but their dependencies on λ are omitted throughout this paper for notational simplicity.

Now choose the grid's λ-divisions to be small enough so that the functions M_i, Q_i, R_i, S_i, and $df^i/d\lambda$ do not change significantly over a single interval. Then, the coefficients of (3.1) are effectively constant. We may also form a single equation from equations (3.1) by taking the projection of the input torque vector $\boldsymbol{u}_i$ onto the velocity vector $df^i/d\lambda$, obtaining the single equation

$$U \equiv u_i \frac{df^i}{d\lambda} = M\dot{\mu} + Q\mu^2 + R\mu + S \tag{3.2}$$

where $M \equiv M_i df^i/d\lambda$, $Q \equiv Q_i df^i/d\lambda$, $R \equiv R_i df^i/d\lambda$, and $S \equiv S_i df^i/d\lambda$. Using the fact that $\mu \equiv \dot{\lambda}$, we may divide (3.2) by μ to obtain

$$\frac{1}{\mu} U = M\frac{\dot{\mu}}{\mu} + Q\mu + R + \frac{1}{\mu} S \tag{3.3}$$

or, using the identity $\dot{\mu}/\mu \equiv (d\mu/dt)/(d\lambda/dt) \equiv d\mu/d\lambda$, we have

$$M\frac{d\mu}{d\lambda} + Q\mu + R + \frac{1}{\mu}(S - U) = 0. \tag{3.4}$$

Note that (3.4) does not explicitly depend on time. Therefore, for purposes of carrying out the dynamic programming algorithm, we may treat the quantities λ and μ as a stage variable and a single-state variable rather than two state variables. Using (3.4) as our (single) dynamic equation, and noting that M, Q, R, and S are approximately constant over one λ-interval, we need to find a solution to (3.2) which meets the boundary conditions

$$\mu(\lambda_k) = \mu_0, \quad \mu(\lambda_{k+1}) = \mu_1 \tag{3.5}$$

in the interval $[\lambda_k, \lambda_{k+1}]$. In order to do this, some form for the inputs $\boldsymbol{u}_i$ 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

$$\boldsymbol{u}_i = Q_i\mu^2 + R_i\mu + V_i \tag{3.6}$$

for the input, where the V_i are constants that may be chosen to make the solution meet the boundary conditions (3.5). Form (3.6) was chosen because it yields particularly simple solutions.

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

A. Case of Noninteracting Torque Bounds

When the joint torque bounds do not interact, the sets E in (2.2) are given by

$$E(q, \dot{q}) = \{(u_1, \cdots, u_n)^T | u^i_{\min}(q, \dot{q}) \le u_i \le u^i_{\max}(q, \dot{q})\}. \tag{3.7}$$

Taking the projection of the input torque vector, as given by (3.6), onto the velocity vector $df^i/d\lambda$ gives $U = Q\mu^2 + R\mu + V$, where $V = V_i df^i/d\lambda$. Plugging this into (3.4) and simplifying gives

$$\frac{d\mu}{d\lambda} = -\frac{1}{\mu}\frac{(S-V)}{M}. \tag{3.8}$$

Solving this equation, we get

$$\lambda = K - \frac{M}{2(S-V)}\mu^2. \tag{3.9}$$

Evaluating the constant of integration K and the constant V so that (3.9) meets the boundary conditions (3.5), one obtains

$$\lambda = \frac{\lambda_k(\mu_1^2 - \mu^2) + \lambda_{k+1}(\mu^2 - \mu_0^2)}{\mu_1^2 - \mu_0^2}. \tag{3.10}$$

Solving for μ in terms of λ gives[4]

$$\mu = \sqrt{\frac{(\lambda_{k+1} - \lambda)\mu_0^2 + (\lambda + \lambda_k)\mu_1^2}{\lambda_{k+1} - \lambda_k}}. \tag{3.11}$$

Now that the path is known over one λ-interval, we need to know the inputs $\boldsymbol{u}_i$ and the components of the incremental cost.

To evaluate the input torques, we may use (3.1) and the value of $\dot{\mu}$. Noting that $\dot{\mu} \equiv \dot{\mu}/\mu \cdot \mu \equiv \mu d\mu/d\lambda$ and using (3.8), we obtain

$$\dot{\mu} = \frac{(V-S)}{M} = \text{constant}. \tag{3.12}$$

The quantities M and S are given, and, using (3.11) and (3.12), V can be calculated to be $V = S + M/2 \cdot (\mu_1^2 - \mu_0^2)/(\lambda_{k+1} - \lambda_k)$, which gives $\dot{\mu} = (\mu_1^2 - \mu_0^2)/2(\lambda_{k+1} - \lambda_k)$. Therefore, the equations for $\boldsymbol{u}_i$ become

$$\boldsymbol{u}_i = Q_i\mu^2 + R_i\mu + S_i + M_i \cdot \frac{\mu_1^2 - \mu_0^2}{2(\lambda_{k+1} - \lambda_k)}. \tag{3.13}$$

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 $\boldsymbol{u}_i$ over the interval $[\lambda_k, \lambda_{k+1}]$ (or equivalently over the interval $[\min(\mu_0, \mu_1), \max(\mu_0, \mu_1)]$ since λ is a monotonic function of μ over the interval under consideration). The maxima/minima may occur at one of three μ values, namely μ_0, μ_1, and the value of μ that maximizes or minimizes $\boldsymbol{u}_i$ over the unrestricted range of μ. In the latter case, the value of μ is $\mu_m = -R_i/2Q_i$. If the condition

$$\min(\mu_0, \mu_1) \le \mu_m \le \max(\mu_0, \mu_1) \tag{3.14}$$

holds, then the point μ_m needs to be tested. Otherwise, the torques must be computed and checked only at the endpoints of the interval.

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

$$C = \int_{\lambda_k}^{\lambda_{k+1}} L(\lambda, \mu, \boldsymbol{u}_i)\, d\lambda \tag{3.15}$$

where μ and $\boldsymbol{u}_i$ are given as functions of λ by formulas (3.11) and (3.13), 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 IV 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 formulas 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 III-C. The algorithm, given the dynamic equations (2.5), the equations of the curve (2.1), the joint torque constraints (3.7), and the incremental cost (3.15), is as follows.

S1: Determine the derivatives $df^i/d\lambda$ of the parametric

[4] In [12] we proved that $\mu \ge 0$ is always true.

functions $f^i(\lambda)$, and from these quantities and the dynamic equations determine the coefficients of (3.1) and (3.2).

S2: Divide the (λ, μ) phase plane into a rectangular grid with N_λ divisions on the λ-axis and N_μ divisions on the μ-axis. Associate with each point (λ_m, μ_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 α to N_λ.

S3: If the column counter α is zero, then stop.

S4: Otherwise, set the current-row counter β to 0.

S5: If $\beta = N_\mu$, go to S12.

S6: Otherwise, set the next-row counter γ to 0.

S7: If $\gamma = N_\mu$, go to S11.

S8: For rows β and γ, generate the curve that connects the $(\alpha - 1, \beta)$ entry to the (α, γ) entry. For this curve, test, as described in the previous paragraphs, to see if the required joint torques are in the range given by inequalities (3.7). 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 (α, γ). 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 (α, γ) that produced the minimum cost, i.e., set $P_{\alpha-1,\beta}$ to γ.

S10: Increment the next-row counter γ and go to S7.

S11: Increment the current-row counter β and go to S5.

S12: Decrement the column counter α 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.

B. 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 (3.7). This assumption is probably correct in many cases, but in others it certainly is not. Here it will be assumed that the inequalities (3.7) are replaced with the constraint (2.2), namely $(u_1, u_2, \cdots, u_n)^T \in E(q, \dot{q})$.

There are a number of situations in which joint torque limits might interact. Consider, for example, a robot that has a common power supply for the servoamplifiers 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 servos. The pump will have finite limits on both the pressure and the volume flow it can produce. Such interacting torque bounds must be considered along with the noninteracting bounds, such as servomotor saturation limits. It is interesting to note that the limits described above all produce set functions E in (2.2) which are *convex*. For example, if the sum of the power consumed (or produced) in all the joints is bounded, one obtains

$$P_{\min} \le u_i \dot{q}^i \le P_{\max}. \tag{3.16}$$

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 hyperrectangular 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 $E(q, \dot{q})$ 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 III-A, it must be possible to determine whether *all* torques are realizable over any given λ-interval. If the torques have the form used in (3.6), 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.

Although 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 (3.6) 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 joint 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 by 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 the following.

S8': For rows β and γ, generate the curve that connects the $(\alpha - 1, \beta)$ entry to the (α, γ) entry. 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.

C. Accommodation of Jerk Constraints

The methods described thus far have ignored the jerk constraints (2.3) 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 *pseudoacceleration* $\dot{\mu}$, say $\nu \equiv \dot{\mu}$. Differentiating the equation for the torque, one obtains

$$\dot{u}_i = \dot{M}_i \nu + M_i \dot{\nu} + \dot{Q}_i \mu^2 + 2Q_i \mu\nu + \dot{R}_i \mu + R_i \nu + \dot{S}_i. \tag{3.17a}$$

Using the identity $d\phi/dt = (d\phi/d\lambda)(d\lambda/dt) = d\phi/d\lambda \cdot \mu$, this equation becomes

$$\dot{u}_i = M_i \dot{\nu} + \left(\frac{dM_i}{d\lambda} + 2Q_i\right)\mu\nu + R_i \nu + \frac{dQ_i}{d\lambda}\mu^3 + \frac{dR_i}{d\lambda}\mu^2 + \frac{dS_i}{d\lambda}\mu. \tag{3.17b}$$

If there are no jerk constraints, then the parameter $\dot{\mu}$ in (3.1) can be manipulated as needed. When there are jerk constraints, we must instead manipulate $\dot{\nu}$ in (3.17b). Equation (3.17b) and constraints (2.3) then give constraints on $\dot{\nu}$, just as (3.1) and constraints (2.2) 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 λ, μ, and ν. 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., (3.6). Because the grid points that the DP algorithm must join from 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.

D. 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_μ rows and N_λ columns. Each entry must contain a cost C and a pointer P. The size of an entry will then be $GS = S_c + S_p$, 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 λ-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

$$TS = PS + N_\lambda N_\mu (S_c + S_p) + N_\lambda S_i + S_d. \tag{3.18}$$

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 manipulator the storage required is, ignoring the program storage

$$TS = 12 N_\lambda N_\mu + 80 + 448 N_\lambda. \tag{3.19}$$

Thus, for a 20 × 80 grid, the storage required is 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_μ points in one column can be connected to each of the N_μ 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. Although 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_μ^2 s, 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, 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 λ-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, so that in practice it may be sufficient to test a single point in the box, say the center, instead of testing all 2^n corners.

IV. 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 [1] 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 λ in this paper. We proved a theorem in [10], 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 in [10] also corrects some minor errors in Bellman's proof.

Like Bellman's proof in [1], we proved 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

$$\frac{d\mu}{d\lambda}=G(\lambda,\ \mu,\ v) \tag{4.1}$$

where the control variable v[5] must meet some set of constraints

$$\Omega_q(\lambda,\ \mu,\ v)\leq 0, \qquad q=1,\ 2,\ \cdots,\ C. \tag{4.2}$$

Also write the objective function J to be maximized as follows:

$$J(v)=\Theta(\mu(\lambda_{\max}))+\int_0^{\lambda_{\max}} F(\lambda,\ \mu,\ v)\ d\lambda \tag{4.3}$$

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

$$\mu_{k+1}=\mu_k+G(\lambda_k,\ \mu_k,\ v_k)\Delta \tag{4.4}$$

$$J(\{v_k\})=\Theta(\mu_N)+\sum_{k=0}^{N-1} F(\lambda_k,\ \mu_k,\ v_k)\Delta \tag{4.5}$$

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

$$\Omega_q(\lambda_k,\ \mu_k,\ v_k)\leq 0, \qquad q=1,\ 2,\ \cdots,\ C. \tag{4.6}$$

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

$$f_n(c)=\sup_{\{v_k\}}\left[\Theta(\mu_N)+\sum_{k=N-n}^{N-1} F(\lambda_k,\mu_k,\ v_k)\Delta\right], \qquad \mu_{N-n}=c. \tag{4.7}$$

Then we have

$$f_0(c)=\Theta(c) \tag{4.8}$$

$$f_{n+1}(c)=\sup_v\ [F(\lambda_{N-n-1},\ c,\ v)\Delta +f_n(c+G(\lambda_{N-n-1},\ c,\ v)\Delta)], \qquad n\geq 0. \tag{4.9}$$

Note that sup has been used instead of max. This is done to allow the use of discontinuous functions and constraint sets which are not closed. 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 2ϵ of the optimum. Since ϵ 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.

We are now in a position to state the main theorem. It states 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 μ can be guaranteed to stay within appropriate bounds.

Theorem: 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 λ and v, for all $\mu_{\min}\leq c_1,\ c_2\leq\mu_{\max}$, and for all λ_1, and λ_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 $d_1,\ d_2,\ \cdots,\ d_{N_d}$. 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 λ_1, λ_2, c_1, c_2, and v. Then the discrete dynamic programming process yields return functions which converge to a limit as the step size Δ goes to zero, provided that μ_k can be guaranteed to stay in the interval $[\mu_{\min},\ \mu_{\max}]$.

The proof of this theorem requires the establishment of four lemmas, but their proofs are not included here due to page limitation. Interested readers are referred to [10].

Roughly speaking, this theorem states that when the functions F and G are continuous of sufficiently high order in μ and piecewise continuous in the stage variable λ, then the discrete dynamic programming process converges.

One important point is that in order to apply the theorem, it must be guaranteed that μ_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 λ, 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 μ greater than some $\epsilon > 0$.

V. 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 first two joints of this same robot were used in [13] to demonstrate the phase plane method for obtaining minimum time trajectories, so a direct comparison of the methods is possible. 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 servodrives of the arm are assumed to consist of voltage source in series with a resistor and an ideal DC motor. This gives the form

$$\begin{aligned}C&=r_tT+r_e\int_0^T \boldsymbol{R}_{ij}\dot{\boldsymbol{q}}^i\dot{\boldsymbol{q}}^j\ dt+r_e\int_0^T\sum_i I_i^2R_i^m\ dt\\&=r_t\int_0^{\lambda_{\max}}\frac{1}{\mu}\ d\lambda+r_e\int_0^{\lambda_{\max}}\mu^2\boldsymbol{R}_{ij}\frac{df^i}{d\lambda}\frac{df^j}{d\lambda}\ d\lambda\\&\quad+r_e\int_0^T\frac{1}{\mu}\sum_i I_i^2R_i^m\ d\lambda\end{aligned} \tag{5.1}$$

[5] This is the same as $\dot{\mu}$ in the previous discussions.

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 $u_i = k_i^m / k_i^g I_i$, where k_i^m and k_i^g are the motor constant and the motor gearing, respectively, the sum in (5.1) can be written as $\Sigma_i (k_i^g)^2 R_i^m/(k_i^m) u_i^2$. The torques u_i are given in terms of λ, μ, and $\dot{\mu}$ by (3.2), which is quadratic in μ, so the integral in (5.1) can be expressed in terms of integrals of powers of μ.

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

$$\Phi_n \equiv \int_{\lambda_k}^{\lambda_{k+1}} \mu^n \, d\lambda = \int_{\lambda_k}^{\lambda_{k+1}} \left[\frac{(\lambda_{k+1}-\lambda)\mu_0^2 + (\lambda-\lambda_k)\mu_1^2}{\lambda_{k+1}-\lambda_k} \right]^{n/2} d\lambda$$

$$= \frac{\lambda_{k+1}-\lambda_k}{\left(1+\frac{n}{2}\right)(\mu_1^2-\mu_0^2)} (\mu_1^{n+2} - \mu_0^{n+2}). \tag{5.2}$$

Equations (5.1), (5.2), and (3.1) give the incremental cost as

$$C = 2r_t\Phi_{-1} + r_e R_{ij} \frac{df^i}{d\lambda}\frac{df^j}{d\lambda}\Phi_1 + r_e \sum_i \frac{(k_i^g)^2 R_i^m}{(k_i^m)^2}$$

$$\cdot [Q_i^2\Phi_3 + 2Q_iR_i\Phi_2 + (R_i^2 + 2Q_i(S_i + M_i\dot{\mu}))\Phi_1$$

$$+ 2R_i(S_i + M_i\dot{\mu})\Phi_0 + (S_i + M_i\dot{\mu})^2\Phi_{-1}]. \tag{5.3}$$

It should be noted that this cost function always has a Φ_{-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 μ is bounded. Thus μ 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 (3.1), we may define a new control variable ρ by the relationship

$$\dot{\mu} = \dot{\mu}_{\min}(\lambda, \mu) + (\dot{\mu}_{\max}(\lambda, \mu) - \dot{\mu}_{\min}(\lambda, \mu))\rho \tag{5.4}$$

so that ρ 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 [14], and are summarized here in Table I. A diagram of the arm is given in Fig. 1. 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 [12]. This robot moves in polar coordinates, i.e., it has a θ joint and an r joint, and was moved from the point (1, 1) to the point (1, −1) along a straight line. The phase plane plots for minimum time, with 10 × 10 and 40 × 40 DP grids are plotted in Fig. 2, 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.

TABLE I
DYNAMIC COEFFICIENTS AND ACTUATOR CHARACTERISTICS FOR PACS ARM

Parameter	Description	Value
τ_θ^{sat}	Saturation torque of θ motor	2.0 Nt.-M.
τ_r^{sat}	Saturation torque of r motor	0.05 Nt.-M.
τ_z^{sat}	Saturation torque of z motor	2.0 Nt.-M.
$V_\theta^{\min}$	Lower voltage limit for θ 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 θ 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_θ^g	Gear ratio for θ 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_θ^m	Motor constant for θ joint	0.0397 Nt.-M./amp
k_r^m	Motor constant for r joint	0.79557×10^{-3} Nt.-M./amp
k_z^m	Motor constant for z joint	0.0397 Nt.-M./amp
R_θ^m	Motor and power supply resistance, θ joint	1 Ω
R_r^m	Motor and power supply resistance, r joint	1 Ω
R_z^m	Motor and power supply resistance, z joint	1 Ω
k_θ	Friction coefficient of θ joint	8.0 Kg.-$M.^2$/sec./rad.
k_r	Friction coefficient of r joint	4.0 Kg./sec.
k_z	Friction coefficient of z joint	1.0 Kg./sec.
M_t	Mass of r joint	10.0 Kg.
M_z	Mass of z joint	40.0 Kg.
J_t	Moment of inertia around θ axis	12.3183 Kg.-$M.^2$
K	Moment of inertia offset term	3.0 Kg.-M.

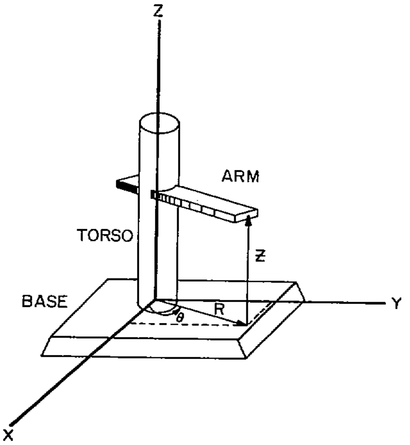


Fig. 1. Diagram of the Bendix PACS robot arm.

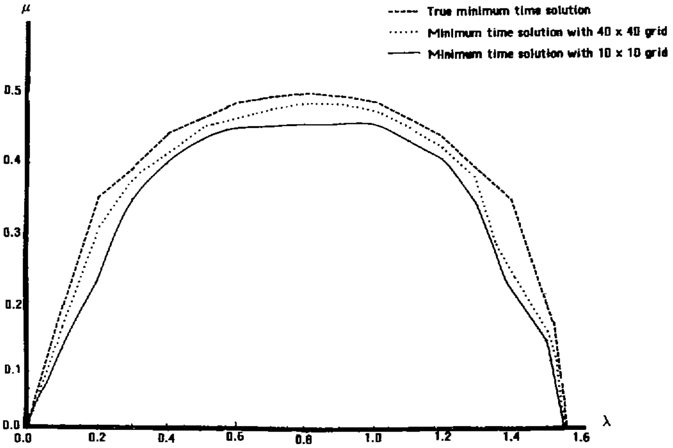


Fig. 2. Minimum time phase plane plots for a polar manipulator with 10 × 10 and 40 × 40 grids.

Fig. 3 shows the phase plane plot for a 10 × 10 grid with a pure minimum time cost function. As can be seen from this figure, joint torque/force versus time plots have close resemblance to motor voltage versus time plots. Hence, joint torque/force versus time

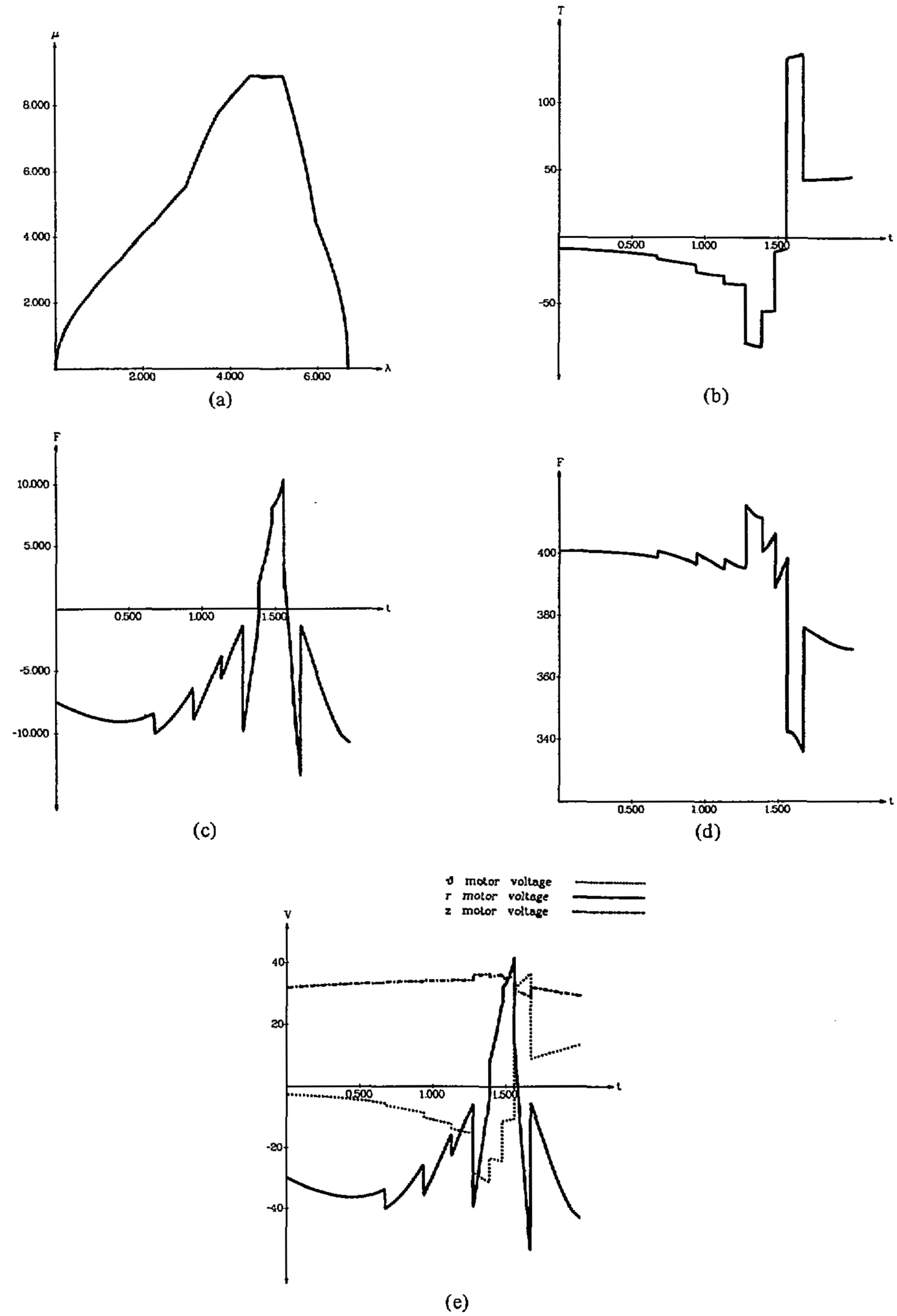


Fig. 3. (a) Minimum time phase plot for PACS arm, 10 × 10 grid. (b) θ joint torque versus time, 10 × 10 grid. (c) r joint force versus time, 10 × 10 grid. (d) z joint force versus time, 10 × 10 grid. (e) Motor voltages versus time, 10 × 10 grid.

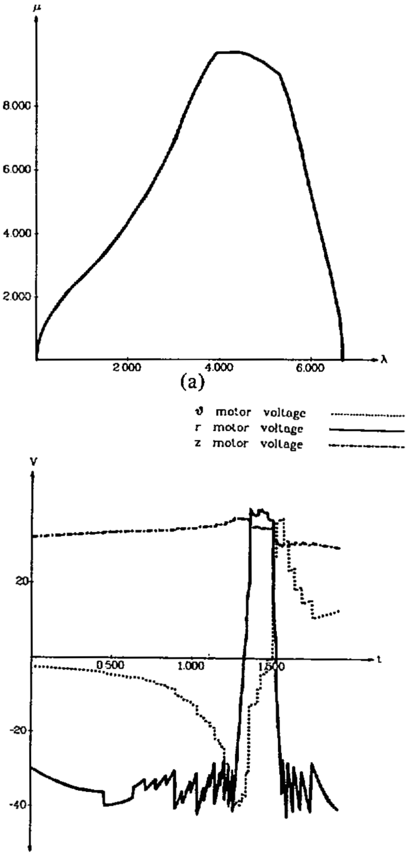
plots will not be included in the remaining figures. Fig. 4 is the same, except that the grid is 40 columns by 160 rows. The calculated traversal times for 10 × 10, 20 × 40, and 40 × 160 grids are 2.000, 1.972, and 1.905 s, respectively, compared to 1.782 s as calculated by the phase plane method.

Figs. 5 and 6 show phase plane plots for a time penalty of 1 unit/s and an energy penalty of 10 units/J. Note that the trajectory is lower than that which is obtained if only time is penalized. The grid sizes are 10 × 10 and 20 × 40.

Figs. 7 and 8 show the results for a minimum time trajectory when, in addition to the torque and voltage constraints given in Table I, the total power sunk or sourced by the robot is limited to 2 kW.

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

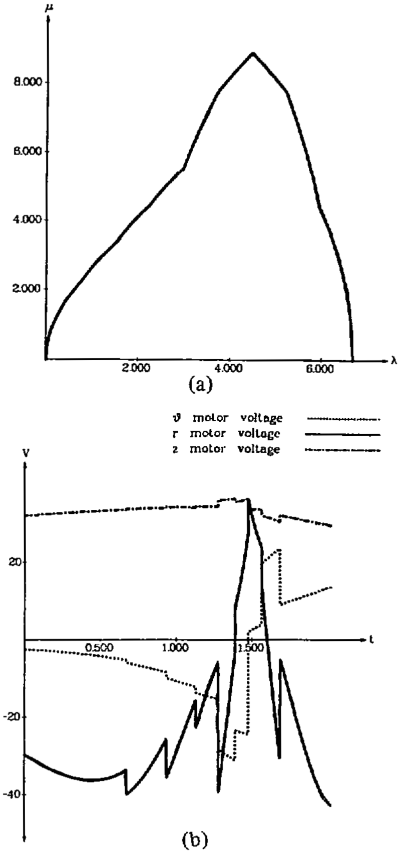
The times given in Table II are real times for the DP algorithm on a lightly loaded system (average of approximately 2 tasks running concurrently). The times must therefore be regarded as approximations to the actual computation times. Table II also lists the function $6 \times 10^{-4} N_\lambda N_\mu^2$, which seems to give a good match



(a)

(b)

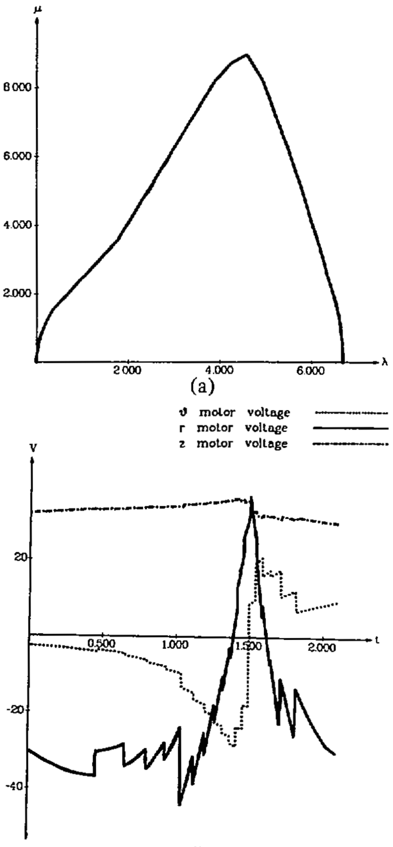
Fig. 4. (a) Minimum time phase plot for PACS arm, 30 × 160 grid. (b) Motor voltages versus time, 40 × 160 grid.



(a)

(b)

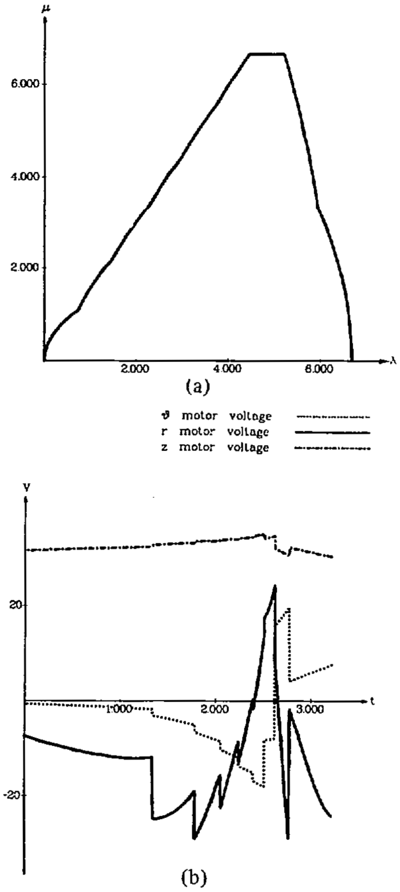
Fig. 5. (a) Phase plot for PACS arm, minimum time-energy, 10 × 10 grid. (b) Motor voltages versus time, 10 × 10 grid.



(a)

(b)

Fig. 6. (a) Phase plot for PACS arm, minimum time-energy, 20 × 40 grid. (b) Motor voltages versus time, 20 × 40 grid.



(a)

(b)

Fig. 7. (a) Minimum time phase plot for PACS arm with power limit, 10 × 10 grid. (b) Motor voltages versus time, 10 × 10 grid.

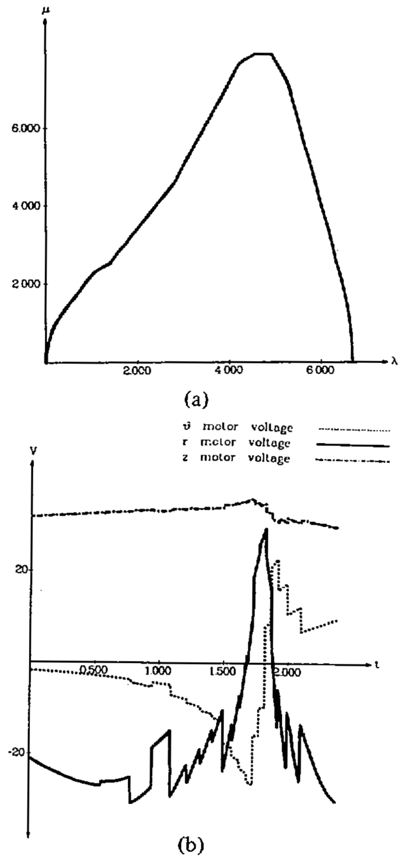


Fig. 8. (a) Minimum time phase plot for PACS arm with power limit, 20 × 40 grid. (b) Motor voltages versus time, 20 × 40 grid.

TABLE II
COMPUTATION TIMES FOR DIFFERENT GRID SIZES (SECONDS)

N_λ	N_μ	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

to the actual running time, as predicted in Section III-D. 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 λ-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 μ-divisions) varies the accuracy of the approximation to the true minimum-cost solution; a finer grid size will in general yield a beter 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 1) choose the λ-divisions to be small enough to make the piecewise-constant dynamic model of the robot sufficiently accurate, 2) choose the μ-divisions to be small enough so that the resulting trajectory is a satisfactory approximation to the true minimum-cost trajectory, and 3) make sure that the μ-divisions are small enough so that the slope of a curve connecting two adjacent points in the grid is small.

VI. Conclusions

We have presented a new, elegant manipulator trajectory planning method for which dynamic programming is used as a main tool. Because of the reduced dimensionality, use of dynamic programming does not suffer from the "curse of dimensionality" and provides power and flexibility to handle very general cost functions and constraints. Moreover, we proved that the DP method produces a solution which converges as the λ interval approaches zero.

The results are significant in that the present method can provide a simple approximation to the truly optimal trajectory solution to any desired degree of accuracy, whereas the conventional methods either treat only special cases or resort to complex approximations or unrealistic assumptions on constraints.

References

[1] R. Bellman, "Functional equations in the theory of dynamic programming—IV, A direct convergence proof," *Annals Math.*, vol. 65, pp. 215-223, Mar. 1957.

[2] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "On the optimal control of robotic manipulators with actuator constraints," in *Proc. 1983 Amer. Contr. Conf.*, June 1983, pp. 782-787.

[3] D. E. Kirk, *Optimal Control Theory: An Introduction.* Englewood Cliffs, NJ: Prentice-Hall, 1971, pp. 53-106.

[4] C.-S. Lin, P.-R. Chang, and J. Y. S. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-28, pp. 1066-1074, Dec. 1983.

[5] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108-119, Feb. 1983.

[6] J. Y. S. Luh and C. E. Campbell, "Collision-free path planning for industrial robots," in *Proc. 21st Conf. Decision Contr.*, Dec. 1982, pp. 84-88.

[7] J. Y. S. Luh and C. S. Lin, "Optimum path planning for mechanical manipulators," *ASME J. Dynam. Syst., Measurement, Contr.*, vol. 102, pp. 142-151, June 1981.

[8] J. Y. S. Luh and M. W. Walker, "Minimum-time along the path for a mechanical arm," in *Proc. 16th Conf. Decision Contr.*, Dec. 1977, pp. 755-759.

[9] L. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-25, pp. 468-474, June 1980.

[10] N. D. McKay, "Minimum-cost control of robotic manipulators with geometric path constraints," Center for Research on Integrated Manufacturing, Univ. Michigan, Robot Syst. Division Tech. Rep. RSD-TR-16-85, Oct. 1985, pp. 94-111.

[11] R. P. C. Paul, *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge, MA: M.I.T. Press, 1981.

[12] K. G. Shin and N. D. McKay, "Minimum-time control of a robotic manipulator with geometric path constraints," in *Proc. 22nd Conf. Decision Contr.*, Dec. 1983, pp. 1449-1457; also in *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 531-541, June 1985.

[13] ——, "Open-loop minimum-time control of mechanical manipulators and its application," in *Proc. 1984 Amer. Contr. Conf.*, San Diego, CA, June 1984, pp. 1231-1236.

[14] ——, "Selection of near minimum-time geometric paths for robotic manipulators," Center for Research on Integrated Manufacturing, Univ. Michigan, Robot Syst. Division Tech. Rep. RSD-14-84, Oct. 1984; and this issue, pp. 501-511.

[15] J. L. Turney, T. N. Mudge, and C. S. G. Lee, "Connection between formulations of robot arm dynamics with applications to simulation and control," Center for Research on Integrated Manufacturing, Univ. Michigan, Robot Syst. Division Tech. Rep. RSD-TR-4-82, Nov. 1982.

[16] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Trans. Man-Machine Syst.*, vol MMS-10, pp. 47-53, June 1969.

Kang G. Shin (S'75–M'78–SM'83), for a photograph and biography, see this issue, p. 511.

Neil D. McKay, for a photograph and biography, see this issue, p. 511.