Proceedings of 23rd Conference
on Decision and Control
Las Vegas, NV, December 1984

ROBOT PATH PLANNING USING DYNAMIC PROGRAMMING[1]

Kang G. Shin and Neil D. McKay

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109

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 the path planning methods in [1],[11], the DP method offers the advantages that it can be used in the general case where (i) the actuator torque limits are dependent on one another, (ii) the cost functions can have an arbitrary form, and (iii) there are constraints on the *jerk*, or derivative of the acceleration.

As a numerical example, the path planning method is simulated for a two-jointed robotic manipulator. The example considers first the minimum-time problem, comparing the solution with that of the phase plane plot method in [11]. Secondly, the sensitivity of the path solutions to the grid size is examined. Finally, the DP method is applied to cases with interactions between joint torque bounds and with cost functions other than minimum-time, demonstrating its power and flexibility.

1. INTRODUCTION

Efficient control of industrial robots is a key to the success of contemporary industrial automation, which is built around 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 *path tracking*. The path tracker is responsible for making the robot's actual position and velocity match desired values of position and velocity [9],[13]; the desired values are provided to the tracker by the path planner. The path planner receives as input a spatial path descriptor [5],[6] from which it calculates a time history of the desired positions and velocities.

Earlier path planners, such as those presented in [7],[8] use linear and/or non-linear 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, or corner, points. Along each segment of the path the (constant) maximum accelerations and velocities are given. Since worst-case bounds for the whole segment must be used, these constant bounds may be quite inaccurate for some parts of the segment. Additionally, these methods provide no rigorous means of obtaining the maximum accelerations and velocities; thus once the path planning process has been completed, the solution must be validated to make certain that the robot's capabilities are not exceeded [2].

The algorithms presented in [1] and [11] 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. These methods use phase plane plots to construct the minimum-time trajectory for a given robot path. It is assumed that the path is given in parameterized form, that the actuator torque limits are functions only of the position and velocity of the manipulator, and that the actuator torque limits are independent of one another. The technique for determining switching points in [11] is different from that used in [1]. 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. However, the method in [11] can handle the general case where the feasible regions in the phase plane are not simply connected, whereas that in [1] cannot.

While these methods are quite elegant, they have three 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 servo amplifiers 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 no 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.

The remainder of this paper is divided into four sections. Section 2 gives a detailed description of the problem. Section 3 presents a solution algorithm using dynamic programming. Section 4 presents numerical examples. Although a simple two-jointed manipulator is used in these examples, it is sufficient to show the significance of the method developed in this paper. Using these numerical examples, first, we compare the results of the direct minimum-time phase plane methods [11] with those of the dynamic programming technique. Secondly, we examine the sensitivity of the path planning solution to the grid size. Third, the dynamic programming method is applied to the general case where (i) cost functions other than minimum time, and (ii) coupling among the actuator torque bounds are considered. Finally, Section 5 states conclusions drawn from our results.

2. PROBLEM STATEMENT

The goal of automation 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 on details of the manufacturing process, and include, in the cases where robots are used, that part of the cost of driving a robot that varies with robot motion, and some maintenance costs. Fixed costs are those that remain constant on a per-unit-time basis, such as taxes and heating costs. If one assumes that the fixed costs dominate, then one obtains the usual assumption, namely that

[1]*The work reported here is supported in part by the U. S. AFOSR Contract No. F49620-82-C-0089 and Robot Systems Division, Center for Robotics and Integrated Manufacturing (CRIM), The University of Michigan, Ann Arbor, Michigan.*

cost per item produced will be proportional to the time taken to produce the item. The minimum time path planning problem is, then, a special case of the minimum cost problem.

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 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 conventional control theory problem, the problem of minimizing the cost of moving a mechanical system 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.[2] 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 <u>two</u> dimensional minimum-cost control problem with some state and input constraints.

More formally, assume that the geometric path is given in the form of a parameterized curve:

$$\mathbf{q}^i = f^i(\lambda), \quad 0 \le \lambda \le \lambda_{\max} \tag{1}$$

where $\mathbf{q}^i$ is the position of the i^{th} joint, $\mathbf{q}=[\mathbf{q}^1,\mathbf{q}^2,\ldots,\mathbf{q}^n]^T$ 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 robot's position and velocity. Then we require that

$$\mathbf{u} = (\mathbf{u}_1,\mathbf{u}_2,\cdots,\mathbf{u}_n)^T \in E(\mathbf{q},\dot{\mathbf{q}}) \tag{2}$$

where $\dot{\mathbf{q}}$ is the first derivative of $\mathbf{q}$ with respect to time, and $\mathbf{u}_i$ is the i^{th} actuator torque/force. E is a function from $R^n \times R^n$ to the space of sets in R^n. The torques $\mathbf{u}_i$ are realizable for position $\mathbf{q}$ and velocity $\dot{\mathbf{q}}$ if and only if the torque vector $\mathbf{u}$ is in the set $E(\mathbf{q},\dot{\mathbf{q}})$.

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:

$$|\dot{\mathbf{u}}_i| \le K_i \tag{3}$$

where K_i is a constant.

The manipulator dynamic equations usually take the form

$$\mathbf{u}_i = \mathbf{J}_{ij}\ddot{\mathbf{q}}^j + \mathbf{C}_{ijk}\dot{\mathbf{q}}^j\dot{\mathbf{q}}^k + \mathbf{R}_{ij}\dot{\mathbf{q}}^j + \mathbf{G}_i \tag{4}$$

where the Einstein summation convention is used, and

$\mathbf{J}_{ij}$ ≡ the inertia matrix
$\mathbf{C}_{ijk}$ ≡ the array of centrifugal and Coriolis coefficients
$\mathbf{R}_{ij}$ ≡ the viscous friction matrix
$\mathbf{G}_i$ ≡ the gravitational loading vector

If the equations of the parameterized path are plugged into these dynamic equations, then they become

$$\mathbf{u}_i = \mathbf{J}_{ij}\frac{df^j}{d\lambda}\dot{\mu} + \left(\mathbf{J}_{ij}\frac{d^2f^j}{d\lambda^2} + \mathbf{C}_{ijk}\frac{df^j}{d\lambda}\frac{df^k}{d\lambda}\right)\mu^2 + \mathbf{R}_{ij}\frac{df^j}{d\lambda} + \mathbf{G}_i \tag{5}$$

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

The cost C will have several components. Obviously, one component will be proportional to traversal time, T. Another component will be proportional to frictional losses. This suggests the form

$$C = r_f T + r_v \int_0^T \mathbf{R}_{ij}\dot{\mathbf{q}}^i\dot{\mathbf{q}}^j dt = r_f \int_0^{\lambda_{\max}} \frac{1}{\mu} d\lambda + r_v \int_0^{\lambda_{\max}} \mu \mathbf{R}_{ij}\frac{df^i}{d\lambda}\frac{df^j}{d\lambda} d\lambda \tag{6}$$

where r_f and r_v are fixed and variable rates of expenditure. Note that although this form bears practical importance and is therefore used throughout this paper, our approach is applicable to *any* form of cost that can be expressed in terms of the time-integral of a function of the robot's states.

The path planning problem then becomes that of minimizing the cost C given by Eq. (6) subject to the (5), the torque constraints (2), and the inequalities Eq. (3).

3. OPTIMIZATION USING DYNAMIC PROGRAMMING

To see how dynamic programming can be applied to this problem, first note that by using the parameterized path (1), the dimensionality of the problem has been reduced; there will be only <u>two</u> 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 the last terms of Eq. (6) are given strictly in terms of the state variables 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 Eqs. (1) and (5).

The informal description given above describes the general approach to the problem. In detail, there are some complications. Therefore, some simplifying but realistic assumptions will be made as we proceed. First, rewrite Eq. (5) in a more convenient form:

$$\mathbf{u}_i = M_i\dot{\mu} + Q_i\mu^2 + R_i\mu + S_i \tag{7}$$

where $M_i \equiv \mathbf{J}_{ij}\frac{df^j}{d\lambda}$, $Q_i \equiv \mathbf{J}_{ij}\frac{d^2f^j}{d\lambda^2} + \mathbf{C}_{ijk}\frac{df^j}{d\lambda}\frac{df^k}{d\lambda}$, $R_i \equiv R_{ij}\frac{df^j}{d\lambda}$, and $S_i \equiv \mathbf{G}_i$. Note that M_i, Q_i, R_i, and S_i are all functions of the parameter λ; 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 $\frac{df^i}{d\lambda}$ do not change significantly over a single interval. Then Eq. (7) effectively has constant coefficients of $\dot{\mu}$, μ^2, and μ. We may also form a single equation from Eq. (7) by taking the projection of the input torque vector $\mathbf{u}_i$ onto the velocity vector $\frac{df^i}{d\lambda}$, obtaining the single equation

$$U \equiv \mathbf{u}_i\frac{df^i}{d\lambda} = M\dot{\mu} + Q\mu^2 + R\mu + S \tag{8}$$

where $M = M_i\frac{df^i}{d\lambda}$, $Q = Q_i\frac{df^i}{d\lambda}$, $R = R_i\frac{df^i}{d\lambda}$, and $S = S_i\frac{df^i}{d\lambda}$. Using the fact that $\mu \equiv \lambda$, we may divide Eq. (8) by μ to obtain

$$\frac{1}{\mu}U = M\frac{\dot{\mu}}{\mu} + Q\mu + R + \frac{1}{\mu}S \tag{9}$$

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

[2] *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 [10]. These joint points are then interpolated to form a geometric path in joint space e.g., [4]. However, this process is not in the scope of the present paper.*

$$M\frac{d\mu}{d\lambda} + Q\mu + R + \frac{\dot{}}{\mu}(S - U) = 0 \tag{10}$$

Using this 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 (8) which meets the boundary conditions

$$\mu(\lambda_k)=\mu_0,\ \mu(\lambda_{k+1})=\mu_1 \tag{11}$$

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

$$\mathbf{u}_i = Q_i\mu^2 + R_i\mu + V_i \tag{12}$$

for the input, where the V_i are constants that may be chosen to make the solution meet the boundary conditions (11). Form (12) 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.

3.1. Case of Non-Interacting Torque Bounds

When the joint torque bounds do not interact, the sets E in (2) are given by

$$E(\mathbf{q},\dot{\mathbf{q}}) = \left\{(\mathbf{u}_1, \cdots, \mathbf{u}_n)^T \,\middle|\, \mathbf{u}^i_{\min}(\mathbf{q},\dot{\mathbf{q}}) \le \mathbf{u}_i \le \mathbf{u}^i_{\max}(\mathbf{q},\dot{\mathbf{q}})\right\} \tag{2a}$$

Taking the projection of the input torque vector, as given by (12), 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. (10) gives

$$\frac{d\mu}{d\lambda} = -\frac{1}{\mu}\frac{(S - V)}{M} \tag{13}$$

Solving this equation, we have $\lambda = K - \frac{M}{2(S-V)}\mu^2$. Evaluating the constant of integration K and the input V so that the boundary conditions (11) are met and solving for μ in terms of λ, one obtains[3]

$$\mu = \sqrt{\frac{(\lambda_{k+1} - \lambda)\mu_0^2 + (\lambda - \lambda_k)\mu_1^2}{\lambda_{k+1} - \lambda_k}} \tag{14}$$

Now that the path is known over one λ-interval, we need to know the inputs $\mathbf{u}_i$ and the components of the incremental cost. The incremental cost has two parts, one proportional to the traversal time and one proportional to the traversal speed. Thus we need the two integrals $\int_{\lambda_k}^{\lambda_{k+1}}\frac{\dot{}}{\mu}d\lambda$ and $\int_{\lambda_k}^{\lambda_{k+1}}\mu d\lambda$. Evaluating the first integral,

$$T = 2\frac{\lambda_{k+1}-\lambda_k}{\mu_1+\mu_0} \tag{15}$$

The second integral is

$$\int_{\lambda_k}^{\lambda_{k+1}}\mu d\lambda = \frac{2(\lambda_{k+1} - \lambda_k)}{3}\left\{\mu_1 + \mu_0 - \frac{\mu_1\mu_0}{\mu_1+\mu_0}\right\} \tag{16}$$

Equations (15), (16), and (7) give the incremental cost as

$$C = \frac{2R\,\tau_v(\lambda_{k+1} - \lambda_k)}{3}\left\{\mu_1 + \mu_0 - \frac{\mu_1\mu_0}{\mu_1+\mu_0}\right\} + 2\tau_f\frac{\lambda_{k+1} - \lambda_k}{\mu_1 + \mu_0} \tag{17}$$

To evaluate the input torques, we may use Eq. (7) and the value of $\dot{\mu}$. Noting that $\dot{\mu} \equiv \frac{\mu}{\lambda}\cdot\dot{\lambda} \equiv \mu\frac{d\mu}{d\lambda}$ and Eq. (13), we obtain

$$\dot{\mu} = \frac{(V-S)}{M} = constant \tag{18}$$

The quantities M and S are given, and, using Eqs. (14) and (18), V can be calculated to be $V = S + \frac{M}{2}\cdot\frac{\mu_1^2 - \mu_0^2}{\lambda_{k+1} - \lambda_k}$, which gives

$$\dot{\mu} = \frac{\mu_1^2 - \mu_0^2}{2(\lambda_{k+1} - \lambda_k)} \tag{19}$$

Therefore, the equations for $\mathbf{u}_i$ become

$$\mathbf{u}_i = Q_i\mu^2 + R_i\mu + S_i + M_i\cdot\frac{\mu_1^2 - \mu_0^2}{2(\lambda_{k+1} - \lambda_k)} \tag{20}$$

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 $\mathbf{u}_i$ over the interval $[\lambda_k,\lambda_{k+1}]$ (or equivalently over the interval $[\mu_0,\mu_1]$ 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 that value of μ that maximizes or minimizes $\mathbf{u}_i$ over the unrestricted range of μ. In the latter case, the value of μ is $\mu_m = -\frac{R_i}{2Q_i}$. If the condition $\min(\mu_0,\mu_1) \le \mu_m \le \max(\mu_0,\mu_1)$ 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.

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 3.3. The algorithm, given the dynamic equations (5), the equations of the curve (1), the joint torque constraints (2), and the coefficients τ_f and τ_v in Eq. (6) 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. (7) and (8).

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 (2). If they are not, go to S10.

S9. Compute the cost of the curve by adding the cost $C_{\gamma\alpha}$ to the incremental cost of joining point $(\alpha-1,\beta)$ to point (α,γ), calculated using Eq. (17). 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.

[3] *In [11] we proved that $\mu \ge 0$ is always true.*

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.

3.2. 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 (2a). This assumption is probably correct in many cases, but in others it certainly is not. Here it will be assumed that the inequalities (2a) are replaced with the constraint (2), namely $(\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n)^T \in E(\mathbf{q}, \dot{\mathbf{q}})$.

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 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. Such interacting torque bounds must be considered along with the non-interacting bounds, such as servo motor saturation limits. It is interesting to note that the limits described above all produce set functions E in Eq. (2) which are *convex*. For example, if the sum of the power consumed (or produced) in all the joints is bounded, one obtains the bounds $P_{\min} \le \mathbf{u}_i \dot{\mathbf{q}}^i \le P_{\max}$. 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 3.1, it must be possible to determine whether *all* torques are realizable over any given λ-interval. If the torques have the form used in Eq. (12), 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. (12) 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 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.

3.3. Accommodation of Jerk Constraints

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

$$\dot{\mathbf{u}}_i = \dot{M}_i \nu + M_i \dot{\nu} + \dot{Q}_i \mu^2 + 2 Q_i \mu \nu + \dot{R}_i \mu + R_i \nu + \dot{S}_i \qquad (21a)$$

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

$$\dot{\mathbf{u}}_i = \frac{dM_i}{d\lambda}\mu\nu + M_i\dot{\nu} + \frac{dQ_i}{d\lambda}\mu^3 + 2Q_i\mu\nu + \frac{dR_i}{d\lambda}\mu^2 + R_i\nu + \frac{dS_i}{d\lambda}\mu \qquad (21b)$$

If there are no jerk constraints, then the parameter $\dot{\mu}$ in Eq. (7) can be manipulated as needed. When there are jerk constraints, we must instead manipulate $\dot{\nu}$ in Eq. (21b). Eq. (21b) and constraints (3) then give constraints on $\dot{\nu}$, just as Eq. (7) and constraints (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" $\dot{\mathbf{u}}_i$, as was done for $\mathbf{u}_i$ when there were no jerk constraints, i.e. Eq. (12). 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.

3.4. Algorithm Complexity

Even though the dimension of the state space for this particular dynamic programming problem is only two, the amount of storage required to implement the method is still a concern. Note, however, that the time required for the algorithm is not a major concern, since the path planning is done off-line.

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. Multiplying this by the number of grid entries and adding the amount of storage PS required for the program gives total storage TS as

$$TS = PS + N_\lambda N_\mu (S_c + S_p)$$

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. 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 $K N_\mu^2$ seconds, where K is a constant. There are N_λ such steps, so the time required is less than $K(N_\lambda - 1)N_\mu^2$.

The dependence of execution time on the number of joints n is much more difficult to assess. The constant K in the equation above depends on both n and the form of the curve to be traversed. The functions M_i and R_i depend on the matrices $\mathbf{J}$ and $\mathbf{R}$ respectively, and the Coriolis term Q_i depends on the three-dimensional array $\mathbf{C}_{ijk}$. 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. However, in practical cases, the number of joints would usually be no more than six, and almost certainly would be less than eight. In any case, these functions only need to be evaluated once per λ-division of the DP grid, and so 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.

4. NUMERICAL EXAMPLES

To demonstrate the use of the dynamic programming algorithm, we present several examples. Although these examples use a simple two degree-of-freedom robot, they are sufficient to demonstrate the method. This same robot was used in [12] 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.

The robot used for these examples is shown in Figure 1. It has one revolute and one prismatic joint, so the robot moves in polar coordinates.

If the path followed is a straight line from $(1,1)$ to $(1,-1)$ in $\mathbf{R}^2$, then the equation of the curve to be followed is $r = \sec\vartheta$, and ϑ ranges from $\frac{\pi}{4}$ to $-\frac{\pi}{4}$. One possible parameterization is $\vartheta = \frac{\pi}{4} - \lambda$, $r = \sec(\frac{\pi}{4} - \lambda)$ where λ ranges from 0 to $\frac{\pi}{2}$. Differentiating these functions with respect to time, using the chain rule to obtain time-derivatives in terms of λ-derivatives, and plugging into the robot dynamic equations gives

$$\mathbf{u}_\vartheta = -\left(J_t - K\sec(\frac{\pi}{4} - \lambda) + M_t \sec^2(\frac{\pi}{4} - \lambda)\right)\dot{\mu} + \mu^2\left(2M_t \sec(\frac{\pi}{4} - \lambda) - K\right)\sec(\frac{\pi}{4} - \lambda)\tan(\frac{\pi}{4} - \lambda) - k_\vartheta \mu \tag{22a}$$

$$\mathbf{u}_r = -M_t \sec(\frac{\pi}{4} - \lambda)\tan(\frac{\pi}{4} - \lambda)\dot{\mu} + \left(2M_t \sec(\frac{\pi}{4} - \lambda)\tan^2(\frac{\pi}{4} - \lambda) - \frac{K}{2}\right)\mu^2 + k_r \sec(\frac{\pi}{4} - \lambda)\tan(\frac{\pi}{4} - \lambda)\mu \tag{22b}$$

See [12] for a detailed derivation of these equations. These equations are Eq. (7) for this particular robot, and are used to calculate the torques required to move from one point in the DP grid to another. The DP algorithm was coded in the C programming language and run under the *UNIX*[4] operating system on a VAX-11/780.

The first examples are the same as those found in [12]; the cost function is strictly minimum time, i.e. $r_f = 1$, $r_v = 0$, and the torque and force bounds are constants for each joint. Figures 2 and 3 show the results of the dynamic programming method when there is no joint friction, plotted on the same axes as the results of the phase plane method. Figure 2 shows the results for a 10×10 grid, Figure 3 is for a 40×40 grid. Figures 4 and 5 show the results when the r joint has a friction coefficient of 15.

Figures 6 and 7 show the optimal trajectory when a combination of minimum time and minimum energy is used as a cost function. The friction coefficient on the r joint is 15, as it was in Figures 4 and 5. However, the cost coefficients r_f and r_v were 1 and 500 instead of 1 and 0. The effect, as would be expected, is to lower the velocity of the optimal trajectory for those parts of the trajectory which require large motions of the r joint. Note that the peak of the curve, which occurs near the midpoint of the trajectory, is at that point for which the velocity of the r joint is zero. This is what would be expected, since frictional losses are zero if the velocity of the r joint is zero. The fixed-cost penalty then dominates, so that a high velocity gives minimum cost.

The apparent saturation of the curve at high velocities is probably caused by discretization effects. If the actual optimal trajectory has a sharp peak, then the DP solution will only ascend into that peak until the squares of the grid are wider than the peak. This would be expected to flatten out the peaks.

Finally, Figure 8 shows the effect of interacting torque bounds. Figure 8 shows a pair of trajectories, one for interacting and one for non-interacting torque bounds. The lower trajectory shows the time-optimal trajectory when the r and ϑ joints are limited to sourcing or sinking a total power of half a Watt each, with each joint independent of the other. The upper trajectory was calculated for interacting torque bounds. In this case, the sum of the absolute values of the powers for the two joints was limited to one Watt. Since power may be distributed as needed in the interacting case, the robot can move faster, even though in both cases the total power consumption is limited to one Watt. The trajectories were calculated with a 40×40 grid, a pure minimum time cost function, and zero friction.

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 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. This in fact happens for the example robot, with parameters set as for Figures 4 and 5, if a DP grid with 40 λ-divisions and only 10 μ-divisions is used. Therefore when choosing grid size, one must (i) choose the λ-divisions to be small enough to make the piecewise-constant dynamic model of the robot sufficiently accurate, (ii) choose the μ-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 μ-divisions are small enough so that the slope of a curve connecting two adjacent points in the grid is small.

[4] *UNIX is a trademark of Bell Laboratories.*

5. CONCLUSIONS

We have presented a new, elegant robot path 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 completely general cost functions and constraints.

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

REFERENCES

[1] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "On the optimal control of robotic manipulators with actuator constraints", *Proc. 1983 American Control Conference*, June 1983, pp. 782-787

[2] J. Hollerbach, "Dynamic scaling of manipulator trajectories", *Proc. 1983 American Control Conference*, June 1983, pp. 752-756.

[3] D. E. Kirk, *Optimal control theory: an introduction*, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 53-106, 1971

[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. Automatic Control*, Vol. AC-28, No. 12, Dec. 1983, pp. 1066-1074.

[5] T. Lozano-Perez, "Spatial planning: A configuration space approach", *IEEE Trans. on Computers*, Vol. C-32, No.2, pp. 108-119, Feb. 1983.

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

[7] J. Y. S. Luh and C. S. Lin, "Optimum path planning for mechanical manipulators", *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol. 102, June 1981, pp. 142-151

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

[9] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "Resolved-acceleration control of mechanical manipulators", *IEEE Trans. Automatic Control*, Vol. AC-25 No. 3, June 1980, pp. 468-474

[10] R. P. C. Paul, *Robot manipulators: Mathematics, programming, and control*, MIT Press, Cambridge, Mass., 1981

[11] K. G. Shin and N. D. McKay, "Minimum-time control of a robotic manipulator with geometric path " constraints", *Proc. 22-nd CDC*, Dec. 1983, pp. 1449-1457.

[12] ——— "Open-Loop Minimum-Time Control of Mechanical Manipulators and Its Application", *Proc. 1984 American Control Conf.*, San Diago, CA, June 1984, pp. 1231-1236.

[13] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses", *IEEE Transactions on Man-Machine Systems*, Vol. MMS-10 No. 2, June 1969, pp. 47-53

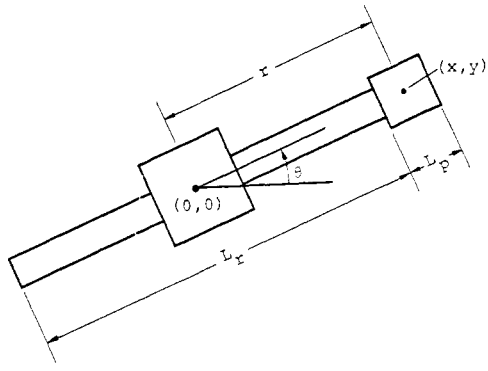

Figure 1. Example of a two degree-of-freedom robot

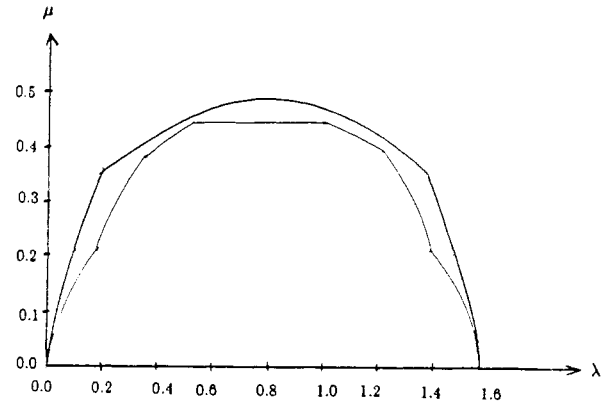

Figure 2. Trajectory for zero friction, minimum time, 10×10 grid Upper curve generated by phase plane method.

Constant	Description	Value
J_θ	Moment of inertia of θ joint	10^{-3} 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} $Kg.$–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 $Kg.$–M/sec^2
u^θ_{max}	Maximum torque on θ joint	1.0 $Kg.$–M^2/sec^2
k_r	Friction coefficient of r joint	0.0 (low friction)
	Friction coefficient of r joint	15.0 (high friction)
k_θ	Friction coefficient of θ joint	0.0

Table 1. Data for the example robot

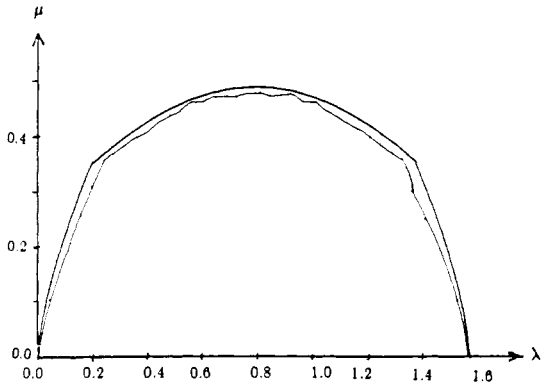

Figure 3. Trajectory for zero friction, minimum time, 40×40 grid
Upper curve generated by phase plane method

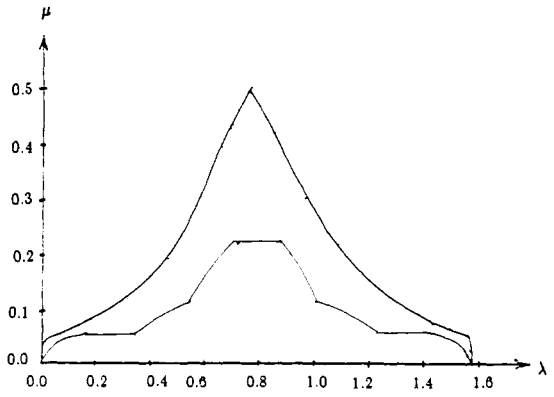

Figure 4. Trajectory for $k_r = 15$, minimum time, 10×10 grid
Upper curve generated by phase plane method

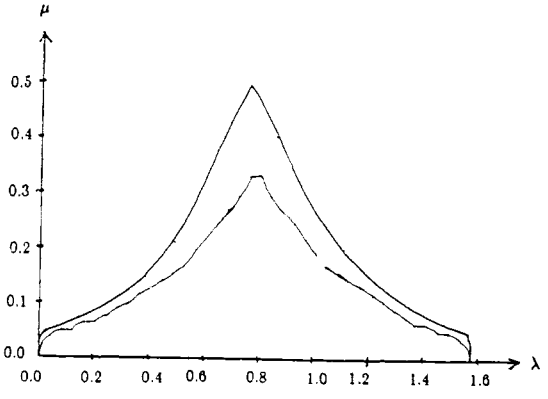

Figure 5. Trajectory for $k_r = 15$, minimum time, 40×40 grid
Upper curve generated by phase plane method

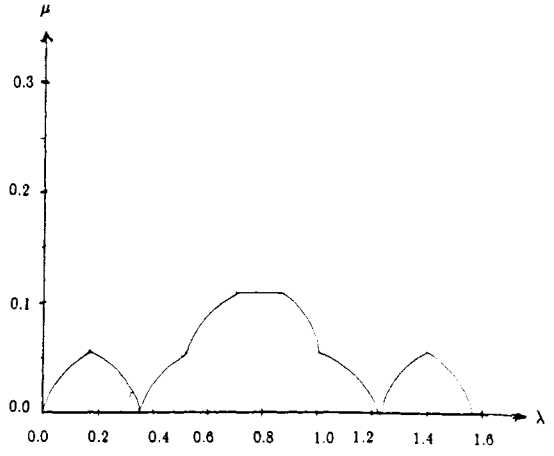

Figure 6. Trajectory for $k_r = 15$, minimum time-energy, 10×10 grid

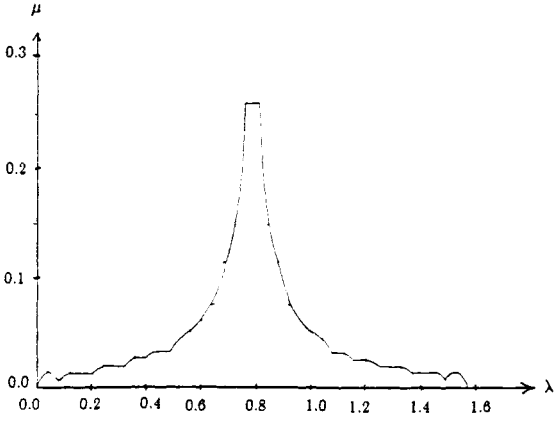

Figure 7. Trajectory for $k_r = 15$, minimum time-energy, 40×80 grid

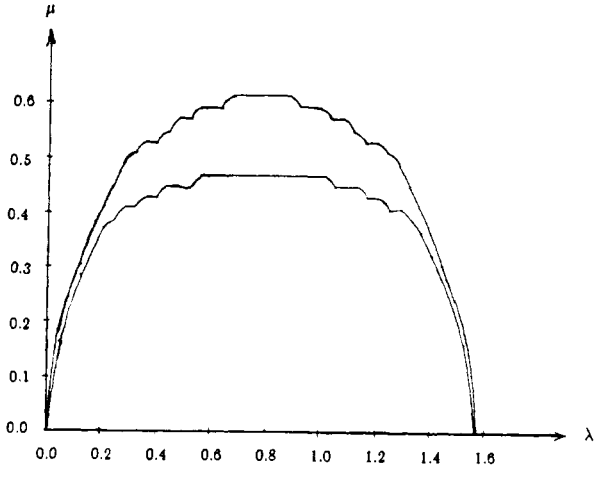

Figure 8. Trajectories for zero friction, minimum time.
Upper curve: maximum power 1 Watt, interacting joints
Lower curve: each joint limited to one-half Watt.