MINIMUM-TIME TRAJECTORY PLANNING FOR INDUSTRIAL ROBOTS WITH GENERAL TORQUE CONSTRAINTS[1]

Kang G. Shin

Neil D. McKay

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

October 1985

CENTER FOR RESEARCH ON INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109-1109

---

[1]This work was supported in part by the NSF Grant No. ECS-8409938 and the US AFOSR Contract No. F49620-82-C-0089. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the funding agencies.

TABLE OF CONTENTS

ABSTRACT

There are a number of trajectory planning algorithms which generate the joint torques/forces required to drive a robot along a given geometric path in minimum or near-minimum time [2, 4, 6, 7, 8, 11]. These methods make fairly specific assumptions about the form of the joint torque/force constraints, thereby limiting their applicability. A method, called the *perturbation trajectory improvement algorithm* (PTIA), is developed here which can generate the joint positions, velocities, and torques required to move a robot along a specified geometric path in minimum time under very general torque constraints.

The PTIA starts with a non-optimal trajectory which meets all the required torque constraints, and perturbs the trajectory in such a way as to always decrease the traversal time for the path. This perturbation process continues until the torque constraints prevent any further improvement in the traversal time. The torque constraints may be expressed in terms of quantities related to torque rather than torque itself; it is possible, for example, to limit velocity, acceleration, jerk, and motor voltage, either singly or in combination. The perturbation trajectory planner also is very simple to implement, and many of the calculations are independent of each other and can therefore be done in parallel. As a demonstrative example, the PTIA is applied to the first three joints of the Bendix PACS Arm.

1. INTRODUCTION

An important goal of contemporary industrial automation with robots is to increase productivity. One way of accomplishing this goal is to move robots as fast as possible, i.e., minimum-time control of robots. However, due to the non-linearity and joint couplings in the robot dynamics, it is possible to obtain only approximate solutions to the problem of direct control of robots in minimum time [3, 5]. To circumvent this difficulty, the control of robots is usually divided into two sequential problems: *off-line trajectory planning* followed by *on-line trajectory tracking.* We shall address in this paper the minimum-time trajectory planning problem.

A number of techniques have been developed for planning minimum-time trajectories of industrial robots [2, 4, 6, 7, 8, 11]. One potential problem with these methods is the assumption that constraints on the torques/forces applied to the robot's joints have specific forms, thereby limiting their applicability. For example, the method described in [7] assumes that the path consists of a sequence of Cartesian straight line segments, and that constant limits on Cartesian velocity and acceleration are known *a priori* along each path segment. Note that it is almost impossible to select such limits without knowing the dynamic properties and the actuator characteristics of the robot. Moreover, since maximum accelerations and velocities are assumed to be constant over some interval, it is necessary to choose them to be the worst case bounds thereby resulting

in under-utilization of the robot's capabilities. To alleviate this problem, we have developed a method in [4] which uses the robot dynamics to obtain approximate velocity and acceleration bounds at each corner or knot point of the robot's path.

More dramatic improvements can be found in [2, 11, 12]. The trajectory planner presented in [2] assumes that joint torque limits are given in terms of the joint's position and velocity, and that the joint torque limits are mutually independent. (Note that the joint actuator limits are dependent on each other when more than one joint actuator share a common power source, e.g., a power supply for DC servo motors and a pump for hydraulic actuators.) A similar trajectory planner described in [11] makes the additional assumption that the velocity dependence of the torque constraints is at most quadratic. Torque constraints which interact, as well as performance indices other than minimum time, can be handled by the trajectory planner described in [12], but the velocity must be discretized rather coarsely if the computations are to be performed in a reasonable amount of time. In practice, torque limits may be relatively complicated, and in fact there may be limits on the derivatives of the torques because of the presence of large motor inductances or because of the compressibility of the hydraulic fluid used in the robot's actuators. It is the goal of this paper to present a minimum-time trajectory planner which is simple and also allows such complicated torque constraints.

This trajectory planner begins with an arbitrary trajectory meeting all the constraints and always alters the trajectory so as to reduce the traversal time

without violating any constraints until a satisfactory solution is obtained. This planner is an iterative algorithm and is called the *Perturbation Trajectory Improvement Algorithm* (PTIA). The method is very easy to use and implement yet very powerful. Also, it is quite different from any known trajectory planners.

The remainder of the paper is divided into four sections. We state the minimum-time trajectory planning problem in the next section. Section 3 describes the PTIA, and gives a simple example of its application. Section 4 analyzes the amount of computation required by the PTIA. In Section 5, the algorithm is applied to the first three joints of a real robot, and some complications regarding the application of jerk constraints are discussed. Section 6 concludes the paper.

2. PROBLEM STATEMENT

The approach to trajectory planning described here assumes, as do the trajectory planners described in [2, 11, 12], that the geometric path to be followed by the robot is given as a parameterized curve in joint space, i.e., the joint positions $\mathbf{q}^i$ are given by

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

where λ is a scalar parameter used to specify geometric paths and n the number of joints that the robot has. It should be noted that in practice the geometric paths are given in Cartesian space. While it is in general difficult to convert a

curve in Cartesian space to that in joint space, it is relatively easy to perform conversion for individual points. One can then pick a sufficiently large number of points on the Cartesian path, convert to points in joint space, and use some sort of interpolation technique (e.g. cubic splines or straight line segments) to obtain a similar path in joint space (see [6] for an example). The resulting joint path is assumed to have the form of Eq. (1).

As in Eq. (1), the parameter λ is sufficient to represent all joint positions, and hence referred to as a *position variable*. The *speed* $\mu \equiv \frac{d\lambda}{dt}$ of the manipulator may be plotted versus the position λ. The speeds and positions of the robot's joints can then be found from the values of λ and μ and the parametric functions f^i. It is well-known that the dynamics of a robot take the general form

$$\mathbf{u}_i = \mathbf{J}_{ij}\ddot{\mathbf{q}}^j + [jk,i]\dot{\mathbf{q}}^j\dot{\mathbf{q}}^k + \mathbf{R}_{ij}\dot{\mathbf{q}}^j + \mathbf{g}_i \tag{2}$$

where $\mathbf{u}_i$ is the i^{th} generalized force, $\mathbf{q}^i$ is the i^{th} generalized coordinate, $\mathbf{J}_{ij}$ is the inertia matrix, the symbol $[jk,i] \equiv \frac{1}{2}\left(\frac{\partial J_{ij}}{\partial \mathbf{q}^k} + \frac{\partial J_{ik}}{\partial \mathbf{q}^j} + \frac{\partial J_{jk}}{\partial \mathbf{q}^i}\right)$ is a Christoffel symbol of the first kind and represents an array of Coriolis coefficients, $\mathbf{R}_{ij}$ is the viscous friction matrix, and $\mathbf{g}_i$ is the gravitational torque vector. By plugging the parametric functions given in Eq. (1) into these dynamic equations (2), joint torques/forces can be calculated in terms of λ, μ, f^i and their derivatives as follows (see [11] for a detailed derivation).

$$\lambda = \mu \tag{3a}$$

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

$$+ \mathbf{C}_{ijk}(\lambda)\frac{df^j}{d\lambda}\frac{df^k}{d\lambda}\mu^2 + \mathbf{R}_{ij}\frac{df^j}{d\lambda}\mu + \mathbf{g}_i(\lambda). \tag{3b}$$

The controller inputs (e.g., voltages or currents) are related to the applied torques/forces, so that control input constraints can also be calculated in terms of these quantities, i.e., λ, μ, f^i and their derivatives.

Also assume that joint torque constraints are expressed in terms of joint positions and velocities,

$$\mathbf{u} \in \mathbf{E}(\mathbf{q},\dot{\mathbf{q}}) \equiv \mathbf{E}(\lambda,\mu) \tag{4a}$$

and the jerk constraints in terms of positions, velocities and accelerations,

$$|\dot{\mathbf{u}}_i| \equiv |\mathbf{F}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})| \equiv |\mathbf{F}(\lambda,\mu,\dot{\mu})| \leq K_i \tag{4b}$$

where $\mathbf{E} : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}^n$ is an admissible input function, $\mathbf{F} : \mathbf{R}^n \times \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}$ is a jerk function, and K_i a constant.

It is further assumed that one phase trajectory (μ vs. λ plot) can be found which meets all constraints. (In practice, a trajectory with zero velocity usually suffices.) This trajectory can then be perturbed to find the one with the shortest traversal time. This perturbation process is made particularly simple by the fact that minimizing time is equivalent to maximizing velocity, i.e.,

$$T \equiv \int_{t_0}^{t_f} dt = \int_0^{\lambda_{max}} \frac{d\lambda}{\mu} \tag{5}$$

subject to the torque constraints (3a), (3b), (4a) and (4b), where $\lambda(t_f)=\lambda_{max}$ is the final position. Here t_0 and t_f denote the initial and final times, where t_f is left free. The minimum-time trajectory planning problem is then to maximize μ, so the phase trajectory should always be pushed upward without violating the torque constraints.

3. PERTURBATION TRAJECTORY IMPROVEMENT ALGORITHM

In practice, a trajectory planner must deal with a variety of arbitrary parametric curves; two representations for curves which immediately suggest themselves are splines and simple sequences of (interpolation) points. We choose to use the latter representation, i.e., the curve (1) is represented as an ordered sequence of points $(\lambda_{(k)},\mathbf{q}_{(k)})$; this proves to be the most natural representation for the application of the PTIA.

The trajectory planning process consists of assigning values of the "velocity" μ and "acceleration" $\dot{\mu}$ at each point. For the sake of simplicity, consider only those constraints which can be expressed in terms of position- and velocity-dependent bounds on the torque (as in Eq. (4a)), i.e., ignore jerk constraints for the time being. Then all constraints can ultimately be given as λ– and μ–dependent constraints on $\dot{\mu}$, or equivalently constraints on $\frac{d\mu}{d\lambda}$ as shown in [11]. In terms of the (λ,μ) plot, each point is assigned a set of allowable

slopes. In fact, having the torque constraint (4a) is equivalent to assigning a pair of vectors to each point in the $\lambda - \mu$ phase plane. One vector represents the maximum possible slope when the system is accelerating (i.e. $\dot{\mu}$ is maximized) and the other represents the slope for deceleration(i.e. $\dot{\mu}$ is minimized). This pair of vectors looks like a pair of scissors, and as the position in the phase plane changes, the angles of both the upper and lower jaws of the pair of scissors change. The phase trajectories must, at every point of the phase plane, point in a direction which lies between the jaws of the scissors. At particular points of the phase plane, though, the jaws of the scissors close completely, allowing only a single value for the slope. At other points the scissors may try to go past the closed position, allowing no trajectory at all. This phenomenon determines the admissible region of the phase plane. Note that the boundary of the admissible region passes through those points which have only a single vector associated with them, corresponding to those states where only a single acceleration value is permitted.

In the discrete approximation, having the maximum and minimum slope sets limits on the differences between the values of μ at adjacent interpolation points. The process of trajectory planning requires that the initial and final points of the curve have zero velocity (or some other fixed velocity) and that the velocities at all the intermediate points be as large as possible, consistent with the slope constraint that the velocities at neighboring points not differ too much.

One approach to the solution of this problem is to try to push the speed higher at each individual point. The value of μ can be pushed higher at each point in succession until none of the velocities can be made any larger. If we call this Algorithm A, then we have

Algorithm A:

A1. Set all velocities to values which are realizable (usually all zeroes).

A2. Push each intermediate point of the curve as high as possible consistent with the slope constraints.

A3. If any of the velocities were changed in step A2, go back to step A2, otherwise exit.

As a practical matter, the search required to find the highest possible velocity in step A2 of Algorithm A may be fairly expensive, especially since it may be repeated many times for a single point. A simpler approach is to just try adding a particular increment to each velocity, and then make the increment smaller on successive passes of the algorithm. This gives

Algorithm A′ *:*

A1'. Set all velocities to values which are realizable (usually all zeroes).

A2'. Set the current increment to some large value.

A3'. Push each intermediate point of the curve up by an amount equal to the current increment, if this is consistent with the slope constraints.

A4'. If any of the velocities were changed in step A3', go back to step A3'.

A5'. If the current increment is smaller than the desired tolerance, stop. Otherwise halve the increment and go to A3'.

Algorithm A' is really just a combination of gradient and binary search techniques. The direction in which the curve must move (i.e. the gradient direction) is known *a priori*, since increasing the velocity always decreases the traversal time, and the amount of the change is successively halved, as in a binary search, until some desired accuracy is achieved. Clearly, this algorithm will terminate in a finite number of steps. Algorithm A' is very simple, except possibly for the slope constraint check required in step A3'. This requires a knowledge of the dynamics and actuator characteristics of the robot. Note that as discussed earlier inadmissible regions are determined by the slope constraints. However, this check is a simple "go/no go" check, and can be isolated as a single function call. (Hereafter this function will be called the *constraint function.*) Hence the trajectory planner can be used with other robots by changing a single, though possibly complicated, function.

Another important characteristic of the constraint function is *locality.* In the case discussed above, the constraints are expressed in terms of λ, μ, and $\frac{d\mu}{d\lambda}$. We need two points to determine the slope $\frac{d\mu}{d\lambda}$, so the constraint depends only upon two points. Therefore when a point of the curve has its μ value changed, it is constrained only by the two adjacent points (due to the slope constraints); the rest of the curve has no influence. This allows much calculation to

proceed in parallel. Step A3' of Algorithm A' can be divided into two sequential steps, one which increments the odd numbered points and one which increments the even numbered ones.[1] Since the even numbered points stay the same while the odd numbered ones are being incremented, and vice versa, the points either side of the incremented points remain stationary, so that the constraint checks are valid. (If *all* points were tested simultaneously, then it is possible, for example, to increment two adjacent points; since in each case the constraint check would be made on the assumption that the other point was remaining stationary, it is possible that the new configuration would not meet the required constraints.)

It is easily seen that the process in Algorithm A' can be extended to more complicated constraints. For example, constraints on the jerk (the derivative of the torque or acceleration) only require a more complicated constraint function, i.e., both Eqs. (4a) and (4b). Of course in this case the constraint function needs three points to calculate second derivatives of the speed. Thus the constraints on a single point will be functions of *two* points either side of the point being checked, rather than one point. This affects the degree of parallelism which can be achieved; step A3' would require three passes instead of two. It also affects the convergence properties of the algorithm, as will be seen later in the examples of Section 5.

As a simple illustration of how the algorithm works, consider a simple one-dimensional problem. Suppose we wish to move an object of mass m from $x=0$

[1]Thus, step A3' requires two passes.

to $x=4$. Further, suppose that there is no friction, and that there are constant bounds on the magnitude of the applied force. There will be only one parametric function f, which may be taken to be the identity function, so that $\lambda=x$. We then have

$$F = m\frac{d^2x}{dt^2} = m\frac{d^2\lambda}{dt^2} = m\frac{d\mu}{dt} = m\frac{d\mu}{d\lambda}\frac{d\lambda}{dt} = m\mu\frac{d\mu}{d\lambda}. \tag{6}$$

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

$$\mu\frac{d\mu}{d\lambda} \approx \mu\frac{\mu_{i+1}-\mu_i}{\lambda_{i+1}-\lambda_i} \approx \frac{\mu_{i+1}+\mu_i}{2}\cdot\frac{\mu_{i+1}-\mu_i}{\lambda_{i+1}-\lambda_i} = \frac{\mu_{i+1}^2-\mu_i^2}{2(\lambda_{i+1}-\lambda_i)}. \tag{7}$$

The torque constraints then become

$$F_{\max} \geq |F| = m|\dot{\mu}| = m\left|\frac{\mu_{i+1}^2-\mu_i^2}{2(\lambda_{i+1}-\lambda_i)}\right|. \tag{8}$$

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

$$|\mu_{i+1}^2 - \mu_i^2| \leq 4. \tag{9}$$

Now consider what happens if Algorithm A is applied. We may look at the intermediate points of the curve in sequence. First, point 1 can be raised by 2, since the adjacent points have μ values of zero (i.e. $\mu_0=\mu_2=0$), and $|2^2 - 0^2| = 4$. Raising the middle point, point 2, we are constrained by the

fact that $\mu_3 = 0$, which limits μ_2 to 2 also. Likewise, we may change μ_3 to 2. This completes step 2 of Algorithm A. Since some of the μ values changed, we try to increase them again. This time only point 2 can be raised, giving a value of $\mu_2 = 2\sqrt{2}$. On the next pass, no μ values change, so Algorithm A terminates. It is easily verified that the solution obtained from Algorithm A is indeed the optimal solution to the discretized problem. (Figure 1 shows the discretized trajectory after passes zero, one, and two of Algorithm A.)

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

4. COMPUTATIONAL REQUIREMENTS AND CONVERGENCE

The PTIA, unlike dynamic programming [12], requires relatively little memory; it requires only one floating point number per interpolation point. However, computation of the CPU time requirements is interesting.

Obviously, the computation time must increase at least linearly with the number of interpolation points on the curve, that is, the size of λ intervals. In fact, the time increases as the square of the number of interpolation points. To see why this is so, consider what happens when the number of interpolation

points is doubled. Since there are twice as many points to check on each pass of the algorithm, the computation time must increase by a factor of two. Recalling that the torque constraints translate into slope constraints, it is clear that the ratio of the amount by which a μ-value may be raised to the distance between λ-values will be approximately constant. Therefore halving the spacing of the interpolation points halves the size of the steps which can be taken in the μ direction, thus doubling the number of steps. This factor of two times the factor of two which results directly from doubling the number of points gives a factor of four increase in computation time. If doubling the number of interpolation points quadruples the computation time, then the time dependence is quadratic in the number of points, i.e., $O(N_\lambda^2)$ where N_λ is the number of the λ intervals.

It is obvious from the discussion that the fineness of the λ intervals will have a significant impact on the running time of the algorithm. It will also affect the accuracy of the results. Similarly to the convergence proof of the trajectory planning with dynamic programming in an extended version[2] of [12] or in [9], we can treat the effect of the grid density on the accuracy of our solution in a quantitative manner.

Bellman proved in [1] that discrete approximations to a continuous optimal control problem will converge as the step size of the dynamic programming 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

[2]This version is currently submitted elsewhere for publication.

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 have proven a theorem in an extended version of [12] and [9] which is an extension of that of Bellman in that it allows the dynamic equation and performance index to be (possibly discontinuous) functions of the stage variable. A similar approach may be applied to the proof of the convergence of our trajectory planning method discussed in this paper. Thus, it is not repeated here.

5. NUMERICAL EXAMPLES

For comparison purposes, we will use the same example robot considered in [12] and [10]. This robot is the Bendix PACS arm, which is cylindrical in configuration and is driven by fixed-field DC motors. (We consider only the first three joints.) The dynamics and actuator characteristics of this robot are given in [10].

First we consider only constraints on joint torques/forces and motor voltages, without considering constraints on their derivatives. The torque and voltage constraints are given in Table 1. The perturbation trajectory planner was written in the C programming language, and run on a VAX-11/780 under the Unix operating system. The planner was tried with a straight-line path, a geodesic in "inertia space" (see [10]), and a joint interpolated path. (The joint-interpolated path has the form $\mathbf{q}^i = \mathbf{q}_s^i + p\,(\mathbf{q}_f^i - \mathbf{q}_s^i)$, where $0 \leq p \leq 1$ and $\mathbf{q}_s^i$ and $\mathbf{q}_f^i$ are the points at which the curve starts and finishes.) The traversal times for these paths are 1.79 seconds, 1.59 seconds and 1.80 seconds respec-

tively. Plots of μ vs. λ, joint positions vs. time, and motor voltage vs. time are shown in Figures 3a through 5c. The traversal times and the various plots are virtually identical to the solutions obtained in [10].

To demonstrate the application of the perturbation technique to problems in which there are constraints on the derivatives of the torques, i.e., jerk constraints (4b), we consider the same problem with the additional constraint that the time derivatives of the joint torques and forces be less than 100, i.e., $K_i = 100$ for all $1 \leq i \leq n$. The time derivatives of the torques are computed using the identity

$$\frac{d\,\mathbf{u}_i}{dt} = \frac{d\,\mathbf{u}_i}{d\lambda}\,\frac{d\lambda}{dt} = \frac{d\,\mathbf{u}_i}{d\lambda}\,\mu. \tag{10}$$

The derivative $\frac{d\,\mathbf{u}_i}{d\lambda}$ was estimated by calculating the difference between the applied torques on successive intervals and dividing by the average of the lengths of the intervals. For a straight-line path with 25 interpolation points, the traversal time is 2.04 seconds. The μ vs. λ plot is shown in Figure 6. For 50 points, the traversal time is 2.26 seconds; the phase plane plot is shown in Figure 7. Note that the trajectory has a "bump" in it; the process has not converged to the proper solution. To understand why this happens, consider the situation shown in Figure 8. The solid line shows the current trajectory, and the dashed lines show what happens when either of the two interior points is raised. In either case, a jerk limit is exceeded, even though the jerk constraint would very possibly be met if *both* points were raised simultaneously. Neither point can

move before the other does, resulting in a sort of "deadlock". Similar situations can occur with longer sequences of points. If jerk constraints are to be included, then obviously we must prevent this sort of situation from occurring. One fix which seems to work in practice is to perform the trajectory planning operation several times with some added constraints, relaxing the constraints each time the trajectory is "improved". The constraints used here were simple velocity limits. On each pass, the velocity limit is raised. If the velocity increment is small enough, the top of the phase trajectory remains flat, and the regions of high inflection which cause the anomalies in the phase trajectory never get a chance to appear. With this modification, a velocity increment of 0.1 at each pass gives the results plotted in Figures 9a through 9c for a straight line with 50 points. 100 points and a velocity increment of 0.025 gives the results plotted in Figures 10a through 10c. The calculated traversal times are 2.03 seconds in both cases.

6. CONCLUSION

A minimum-time trajectory planning scheme has been presented which (i) is extremely simple for actual implementation, and (ii) allows the use of very general types of torque constraints. This trajectory planner has been shown to give the same results as the trajectory planner described in [11] for the paths and torque constraints given in [10]. In addition, this scheme has been applied to a case where there are limits on jerk as well as torque.

Since the PTIA is simple but general enough to accommodate various realistic constraints, it has high potential use for automatically generating trajectories of the growing number of industrial robots.

REFERENCES

[1] R. Bellman, "Functional Equations in the Theory of Dynamic Programming - VI, A direct Convergence Proof," *Annals of Mathematics,* vo. 65, no. 2, pp. 215-223, March 1957.

[2] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "On the Optimal Control of Robotic Manipulators with Actuator Constraints," *Proceedings of the 1983 Automatic Control Conference,*

[3] M. E. Kahn and B. Roth, "The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains," *ASME Journal of Dynamic Systems, Measurement, and Control,* pp. 164-172, September 1971.

[4] B. K. Kim and K. G. Shin, "Minimum-time path planning for robot arms and their dynamics," *IEEE Trans. System, Man, and Cybernetics,* vol. SMC-15, no. 2, pp. 213-223, March/April 1985.

[5] B. K. Kim and K. G. Shin, "Suboptimal Control of Industrial Manipulators with a Weighted Minimum Time-Fuel Criterion," *IEEE Trans. on Automatic Control,* vol. AC-30, no. 1, pp. 1-10, January 1985.

[6] C.-S. Lin, P.-R. Chang, and J. Y. S. Luh, "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Mechanical Manipulators," *IEEE Trans. on Automatic Control,* vo. AC-28, no. 12, pp. 1066-1074, December 1983.

[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, pp. 142-151, June 1981.

[8] J. Y. S. Luh and M. W. Walker, "Minimum-Time Along the Path for a Mechanical Arm," *Proceedings of the IEEE Conference on Decision and Control,* pp. 755-759, December 1977.

[9] N. D. McKay, "Minimum-Cost Control of Robotic Manipulators with Geometric Path Constraints," Ph.D. thesis, The University of Michigan, September 1985.

[10] K. G. Shin and N. D. McKay, "Selection of Near Minimum-Time Geometric Paths for Robotic Manipulators," *Proceedings of the 1985 ACC,* pp. 346-355, June 1985. (Also to appear in *IEE Trans. on Automatic Control).*

[11] K. G. Shin and N. D. McKay, "Minimum-Time Control of a Robotic Manipulator with Geometric Path Constraints," *IEEE Transactions on Automatic Control,* vol. AC-30, no. 6, pp. 531-541, June 1985.

[12] K. G. Shin and N. D. McKay, "Robot Path Planning Using Dynamic Programming," *Proceedings of the 23rd CDC,* pp. 1629-1635, Dec. 1984.

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_θ^{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_θ^{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./sec.
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.

Table 1. Dynamic coefficients and actuator characteristics for PACS arm.

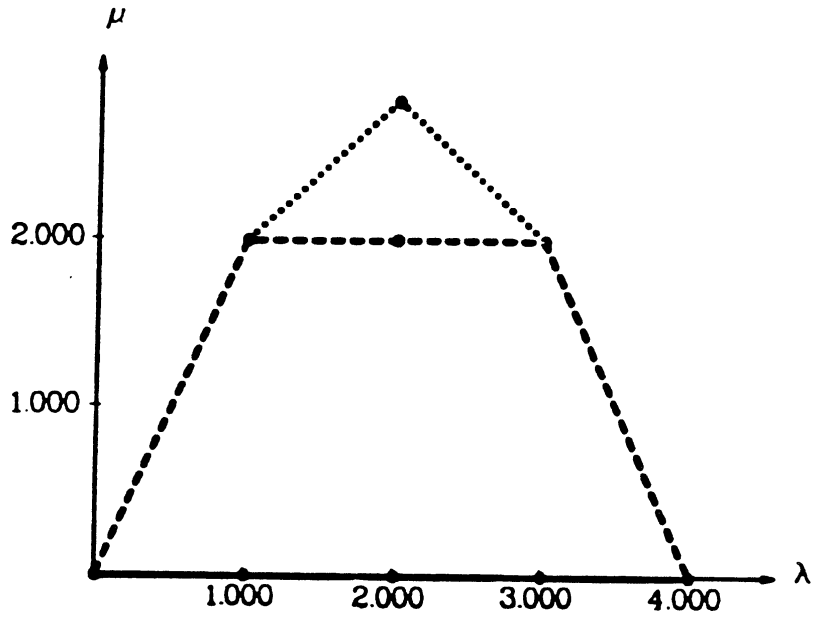

Figure 1. Results of Algorithm A.

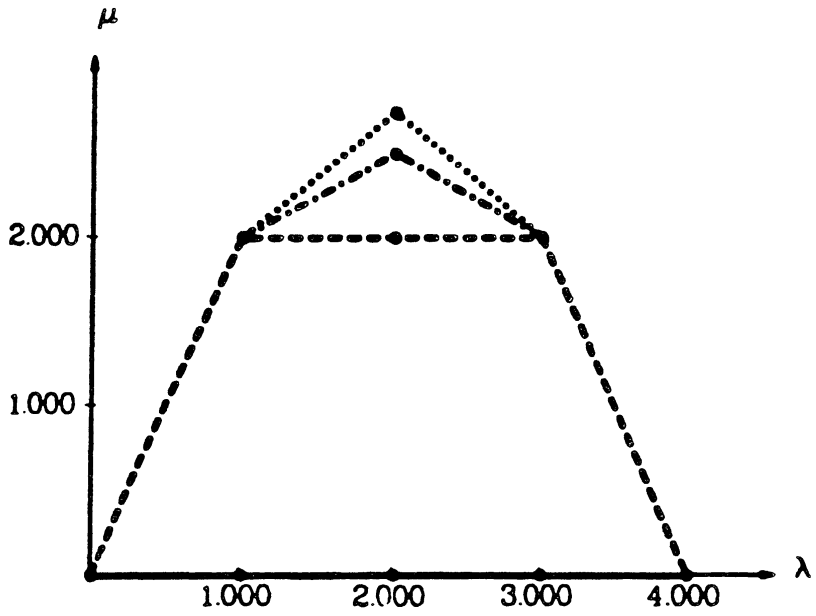

Figure 2. Results of Algorithm A'.

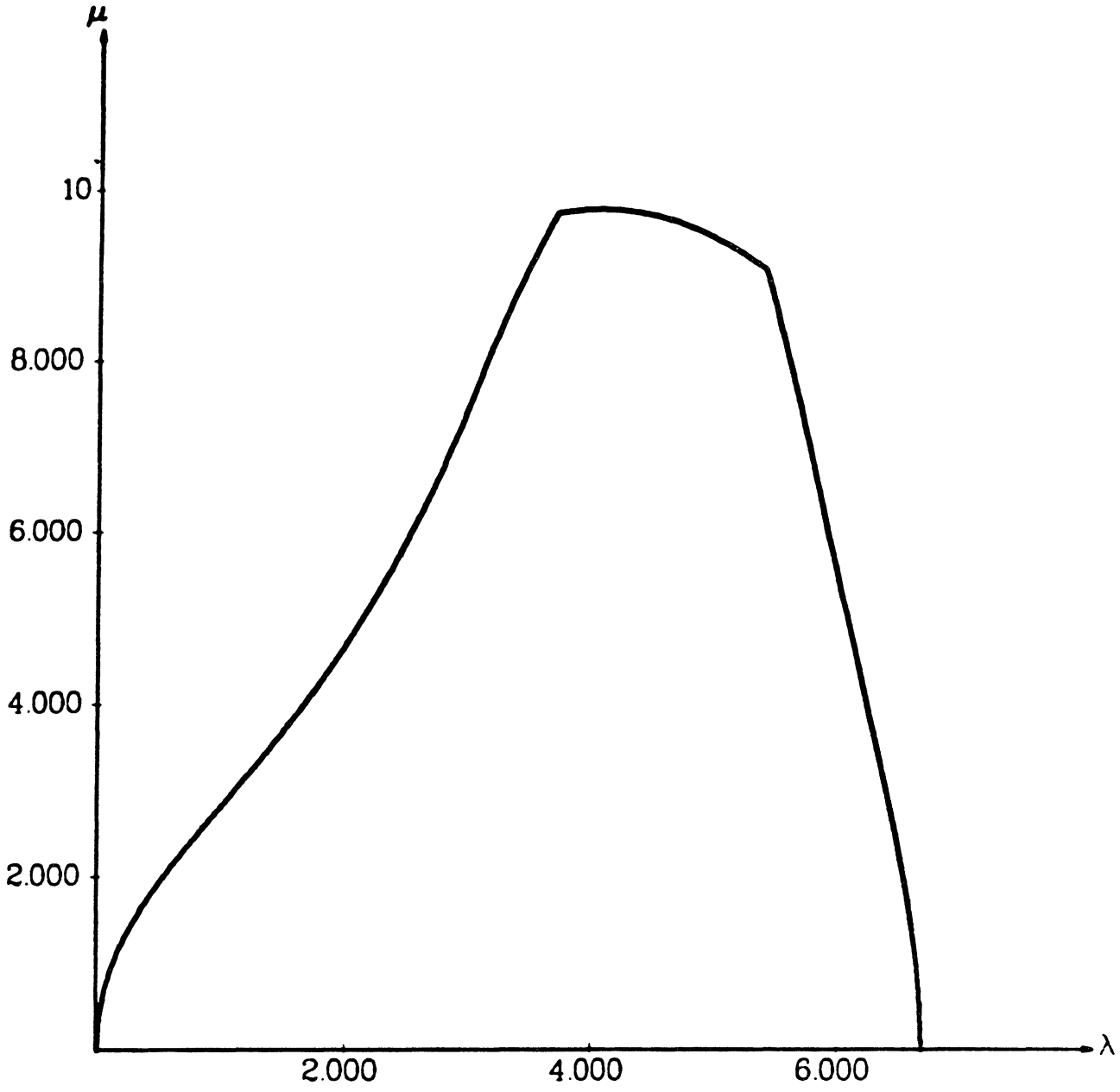

Figure 3a. Phase plane plot for straight line.

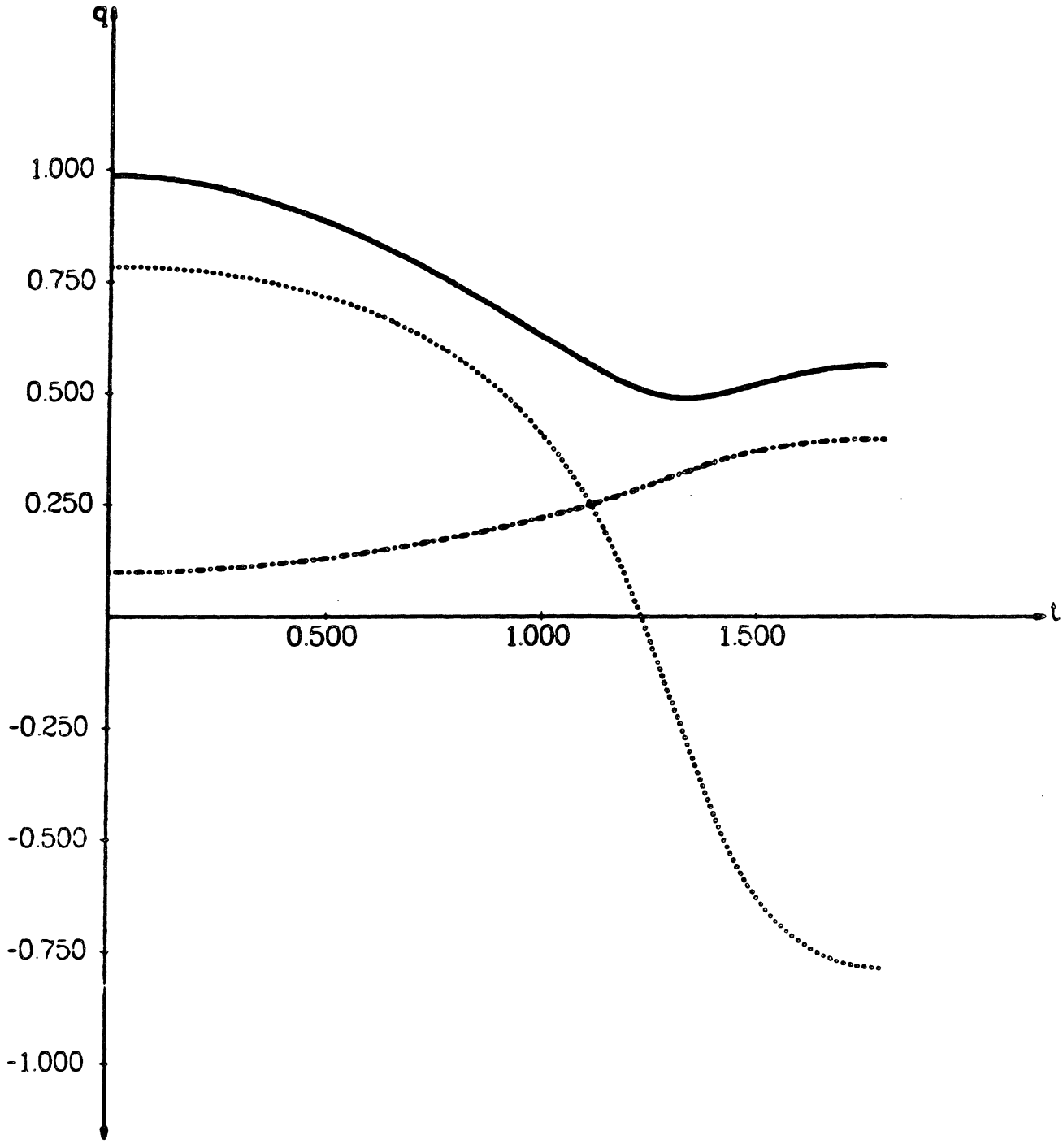

Figure 3b. Joint position vs. time for straight line.

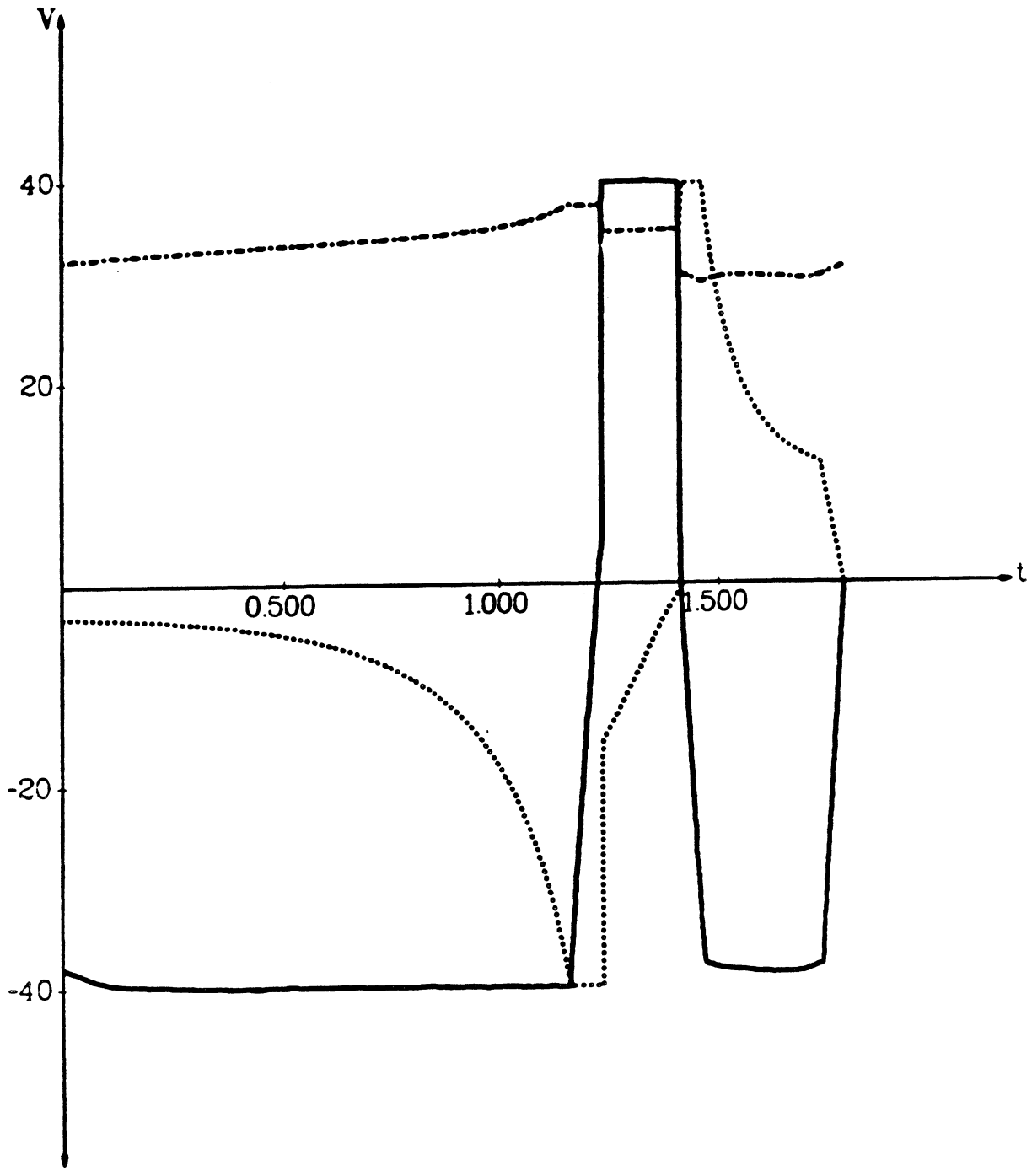

Figure 3c. Motor voltage vs. time for straight line.

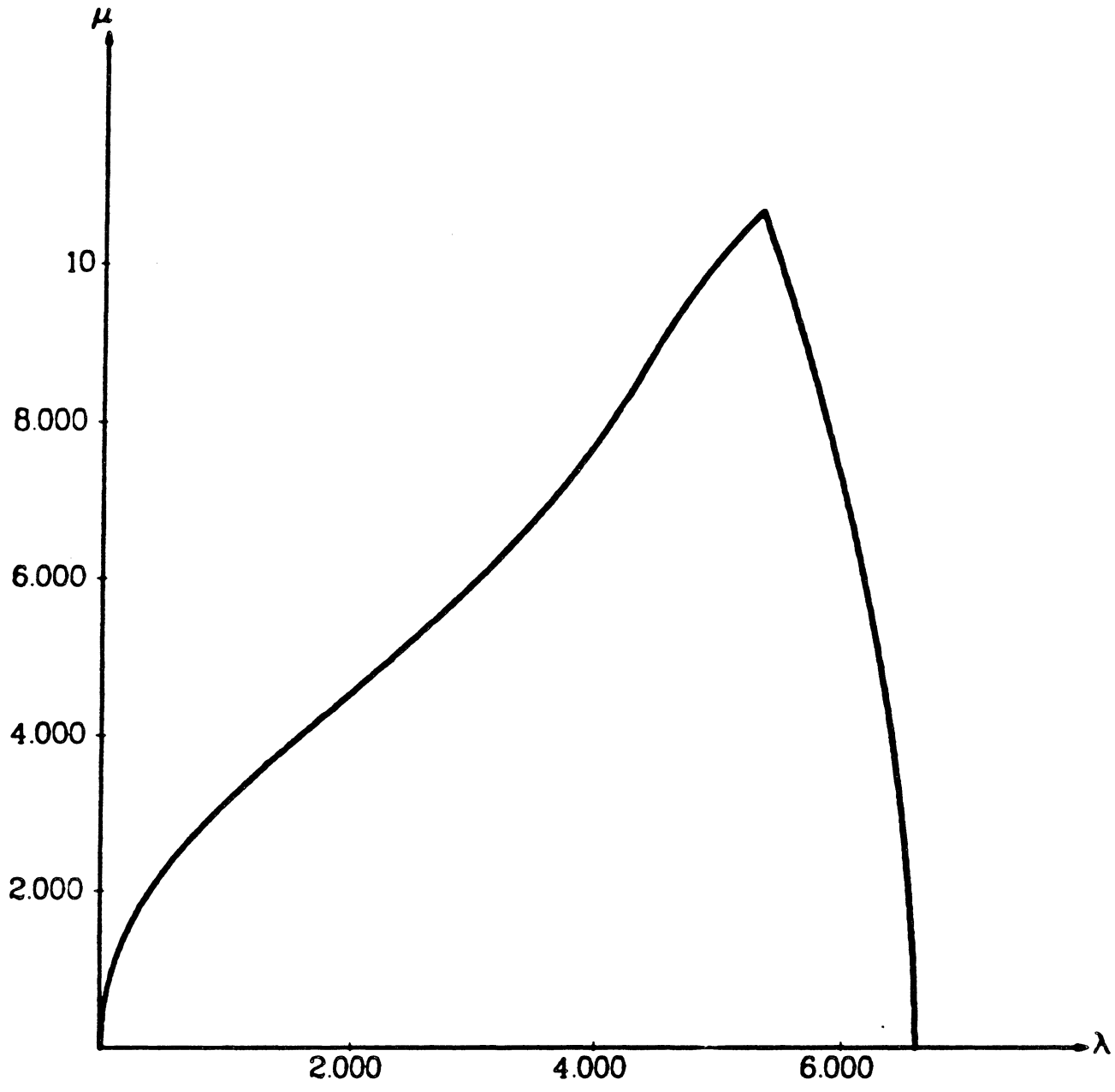

Figure 4a. Phase plane plot for geodesic.

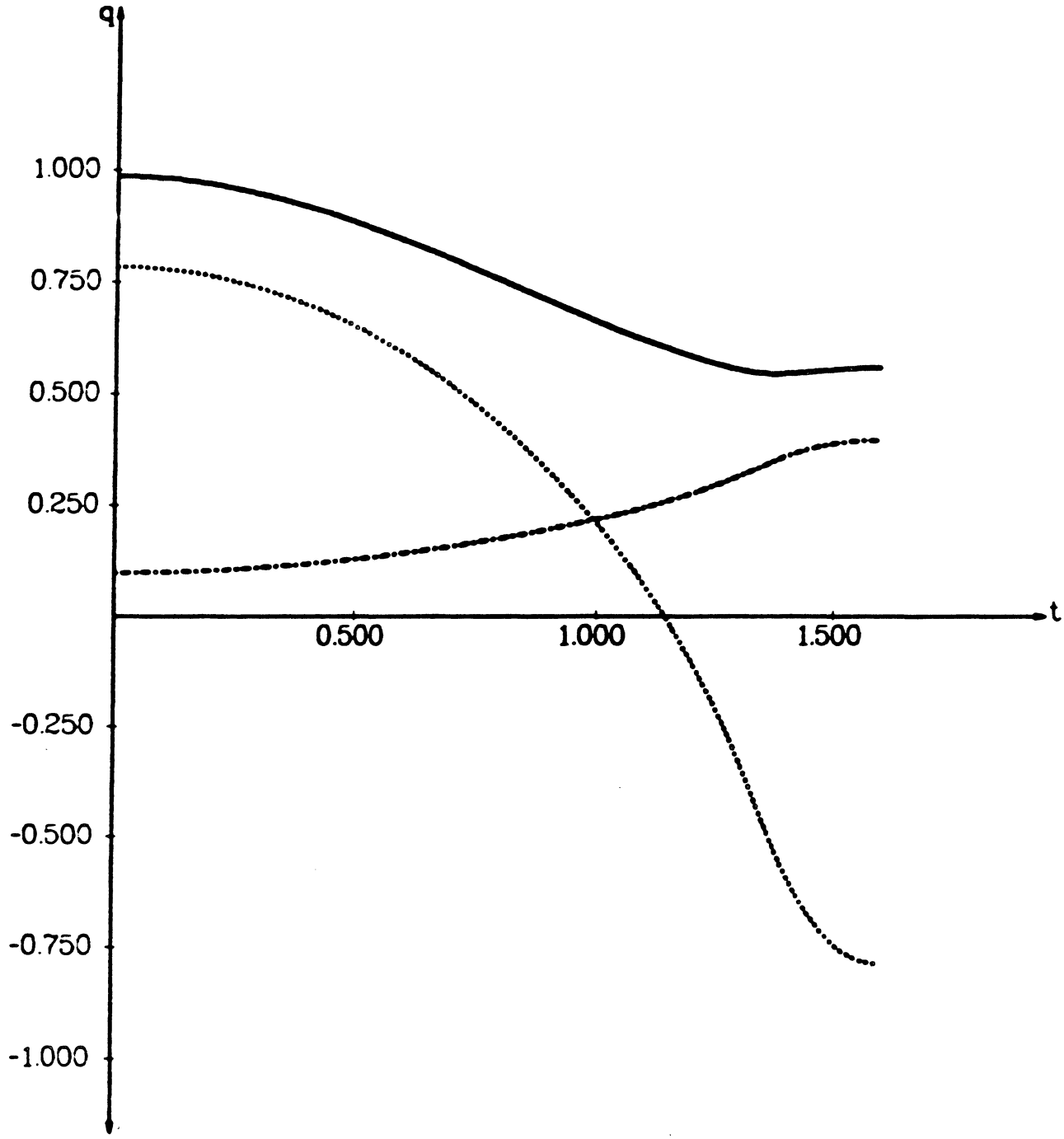

Figure 4b. Joint position vs. time for geodesic.

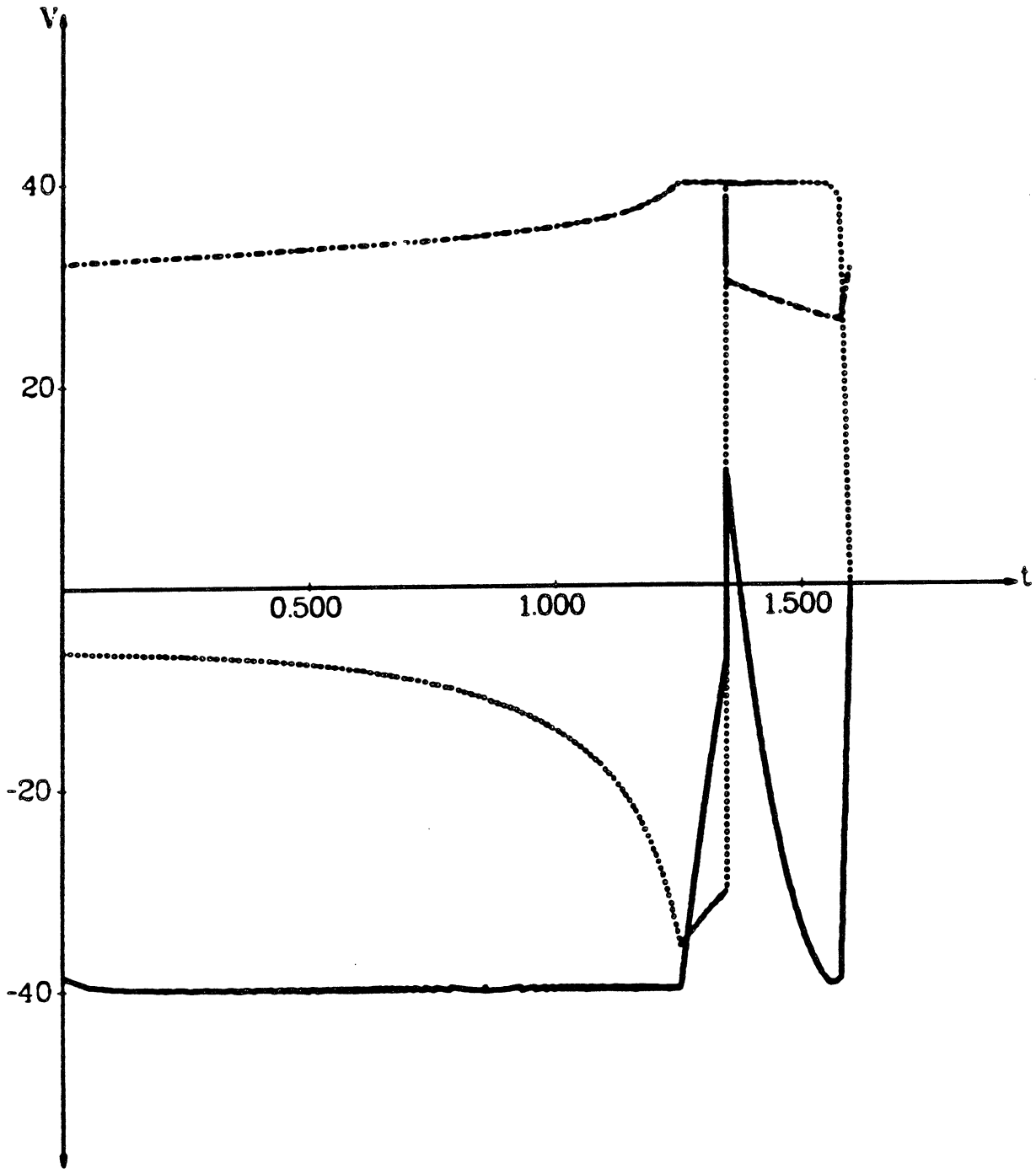

Figure 4c. Motor voltage vs. time for geodesic.

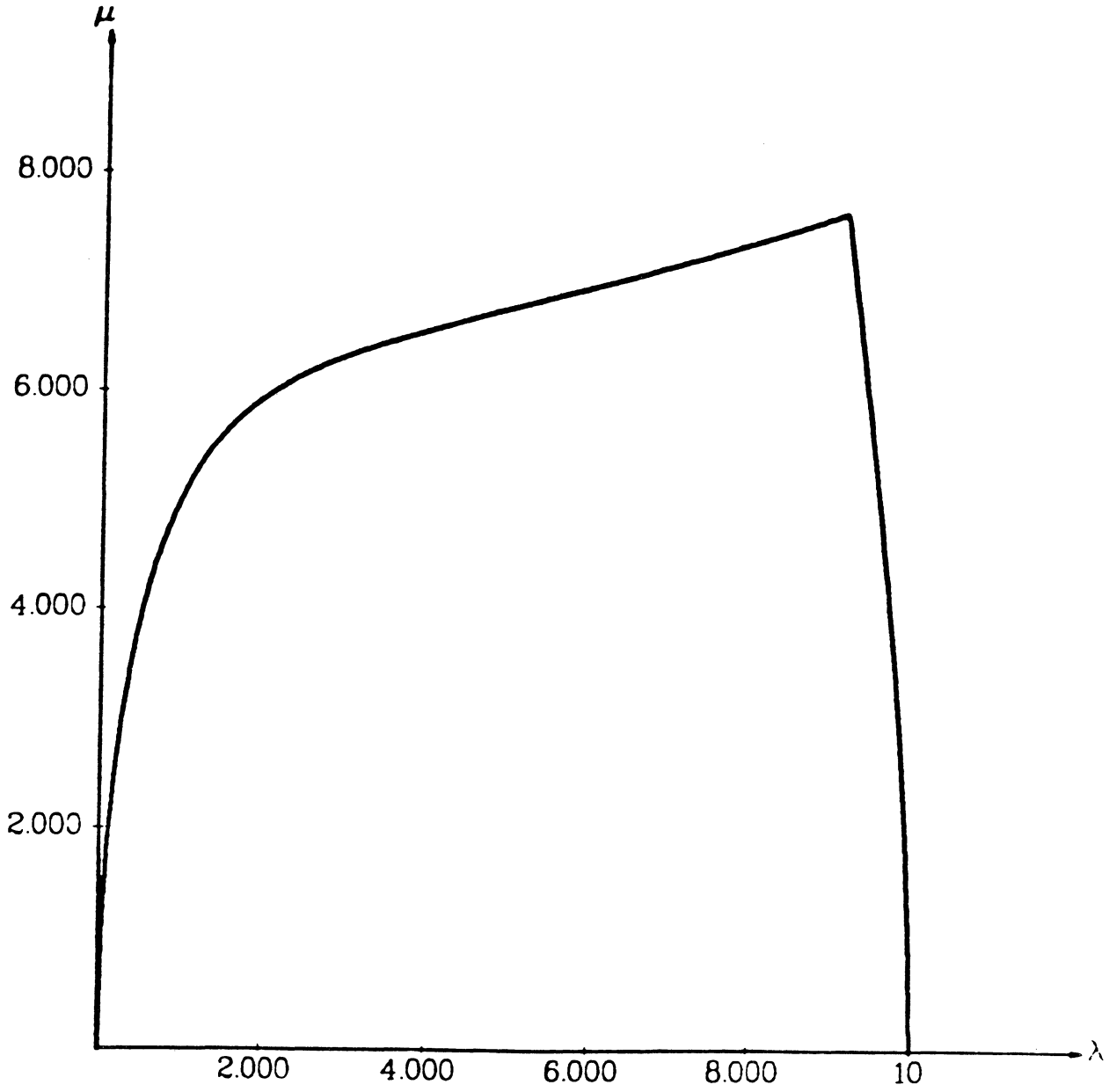

Figure 5a. Phase plane plot for joint-interpolated curve.

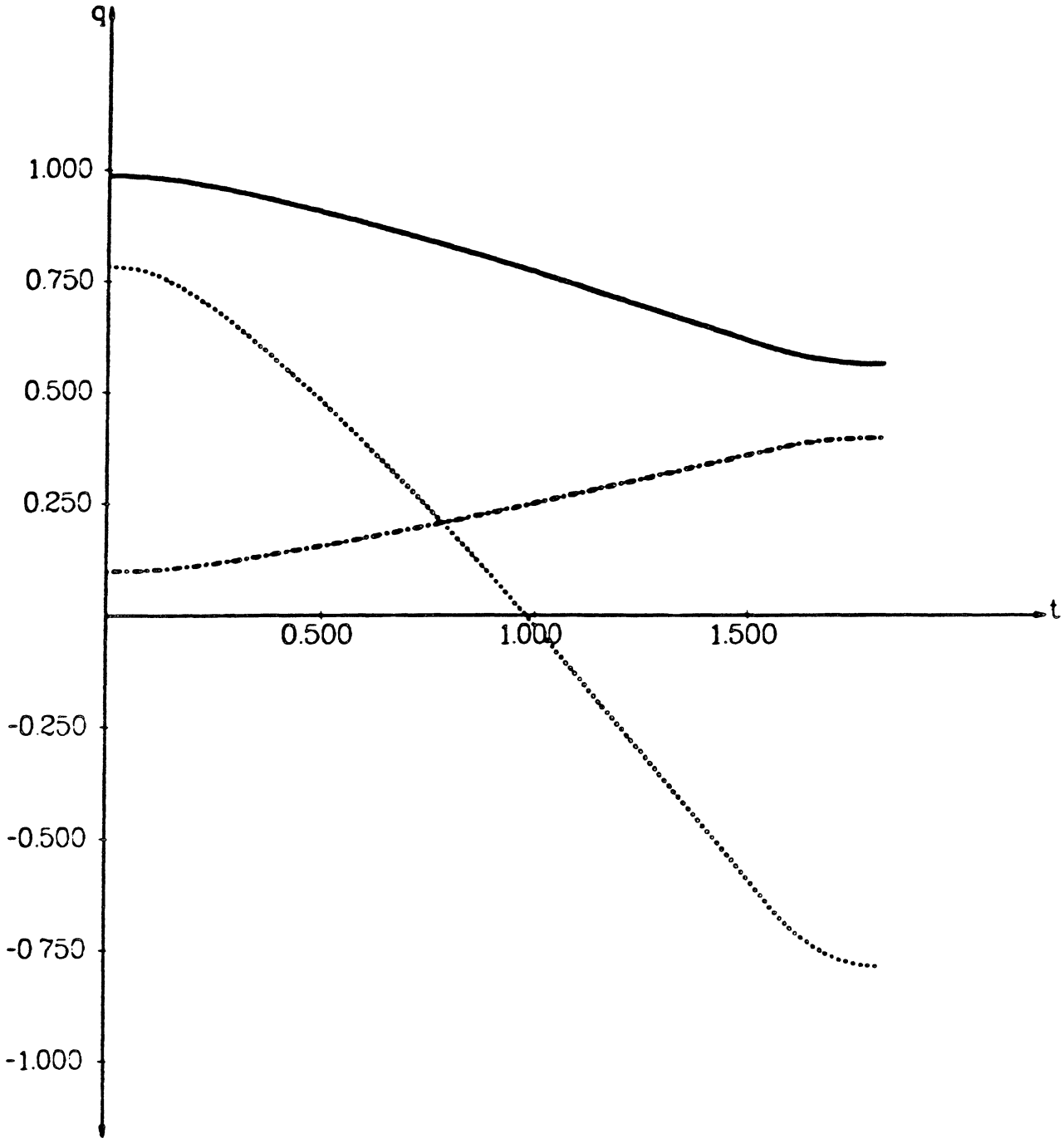

Figure 5b. Joint position vs. time for joint-interpolated curve.

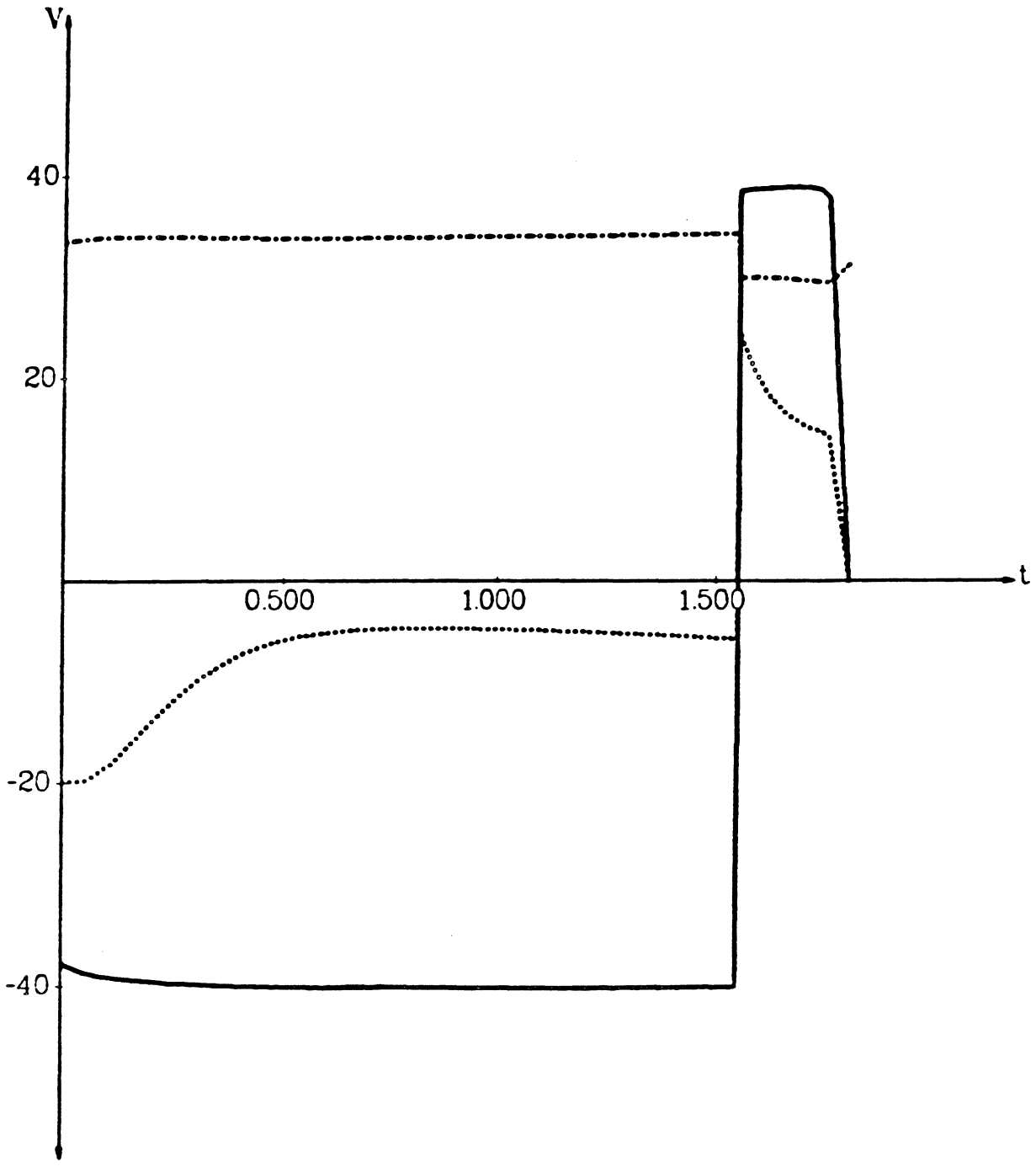

Figure 5c. Motor voltage vs. time for joint-interpolated curve.

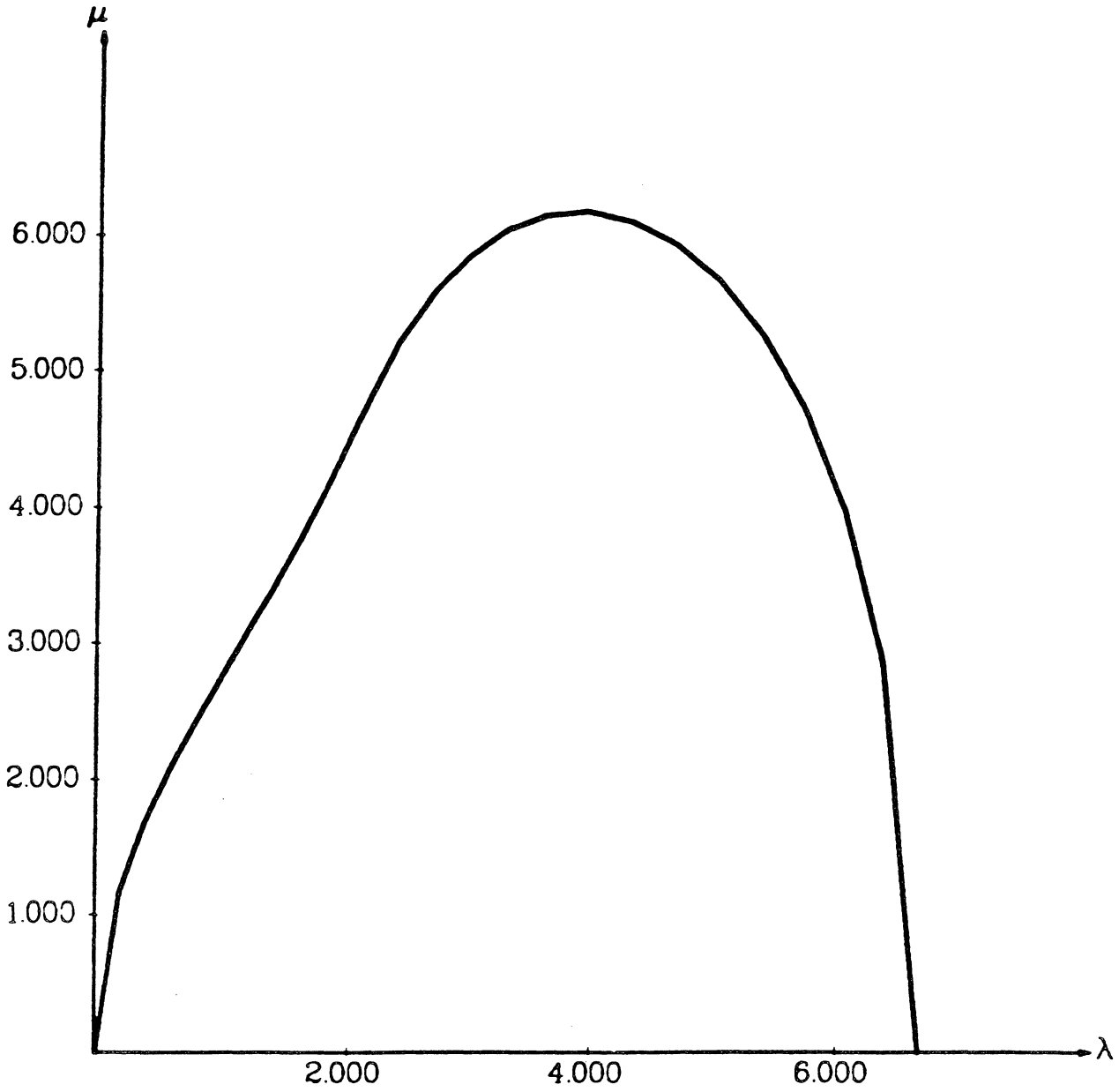

Figure 6. Phase plane plot for straight line. 25 Interpolation points, with jerk constraints.

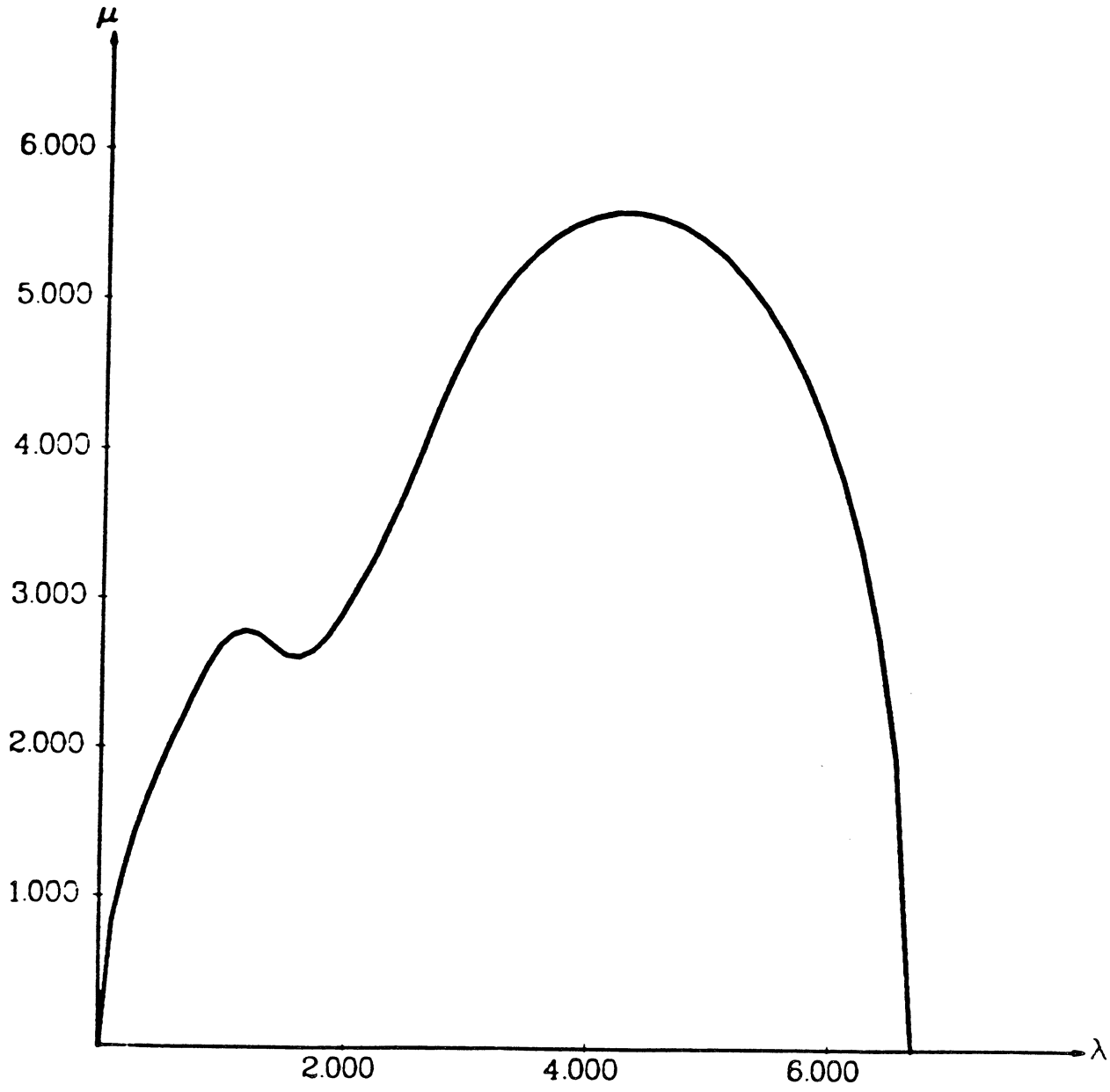

Figure 7. Phase plane plot for straight line, 50 interpolation points, with jerk constraints.

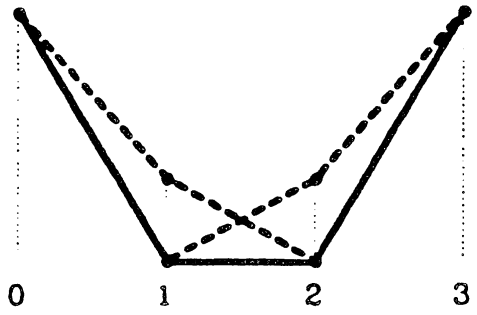

Figure 8. Illustration of "deadlock" which may occur when jerk constraints are applied.

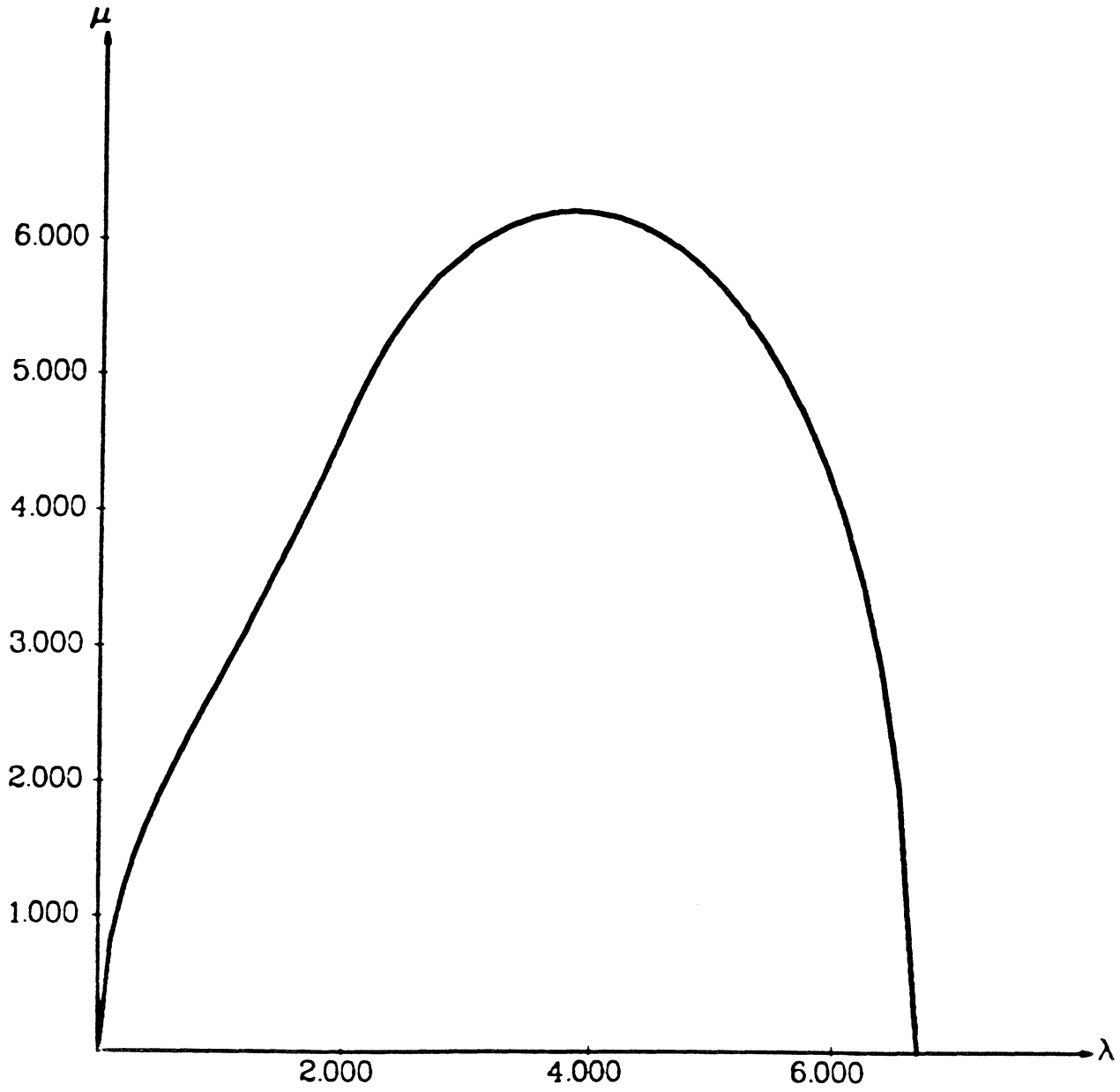

Figure 9a. Phase plane plot for straight line. 50 Interpolation points, velocity increment 0.1.

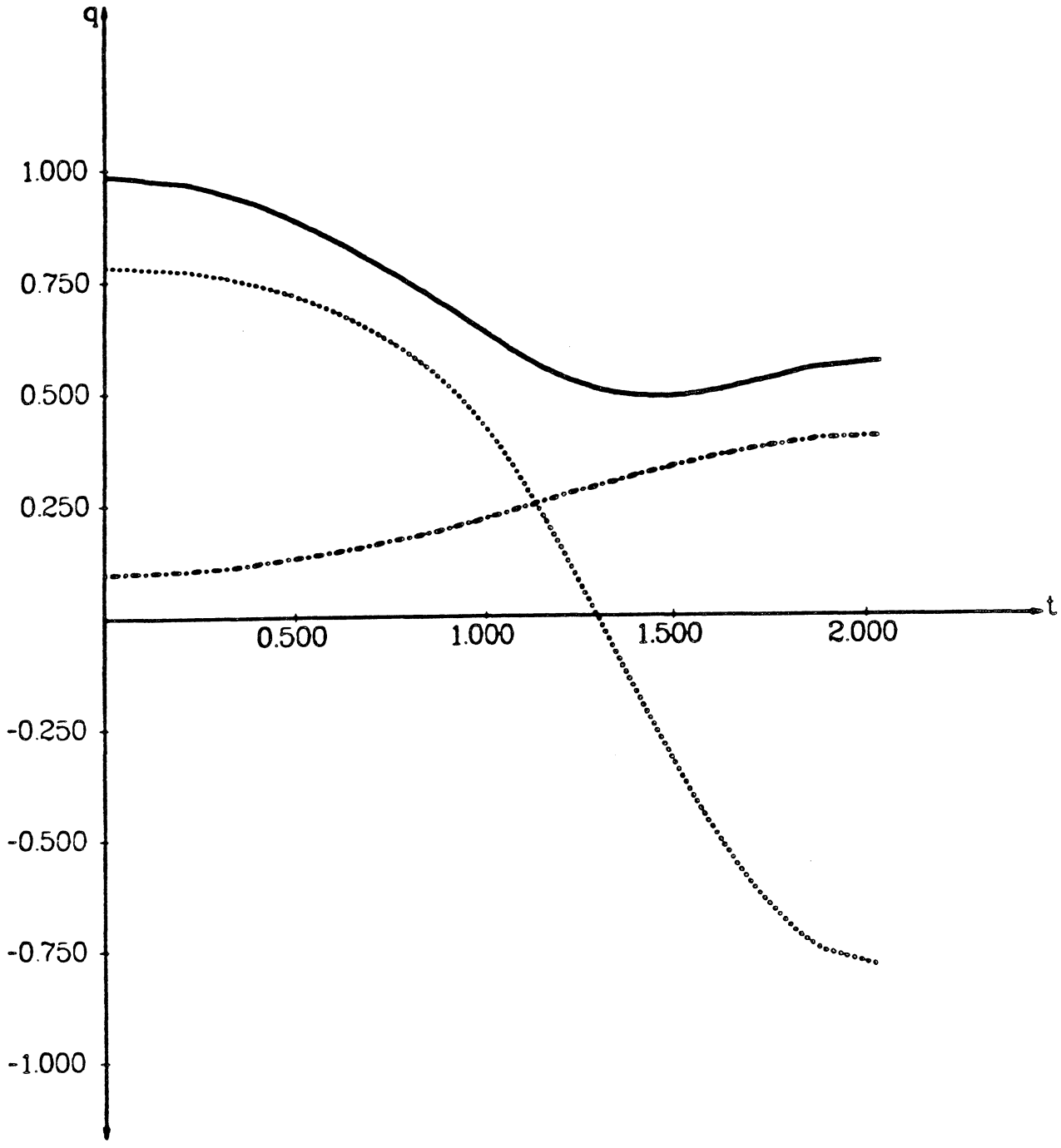

Figure 9b. Position vs. time for straight line. 50 Interpolation points, velocity increment 0.1.

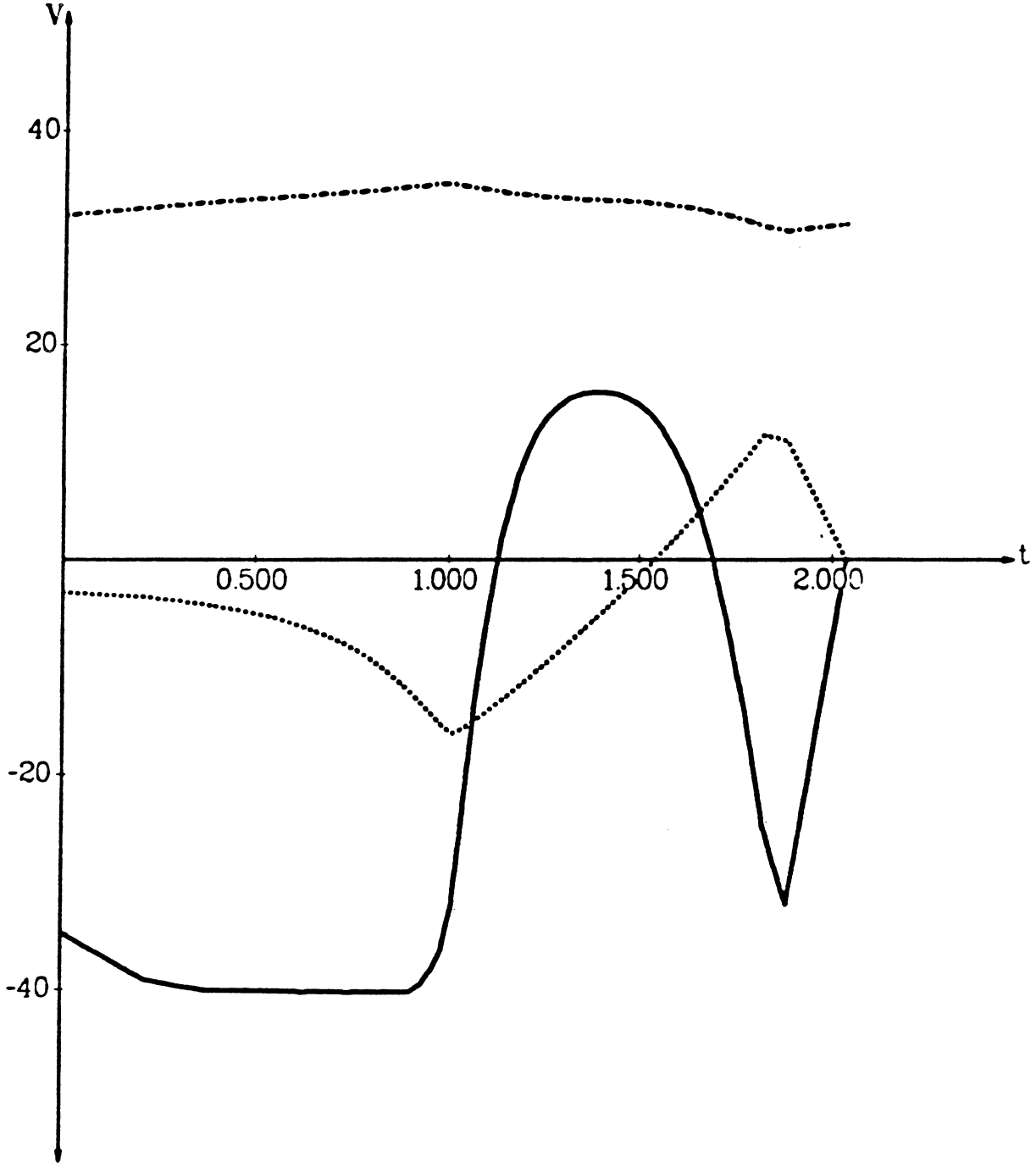

Figure 9c. Motor voltage vs. time for straight line. 50 Interpolation points, velocity increment 0.1.

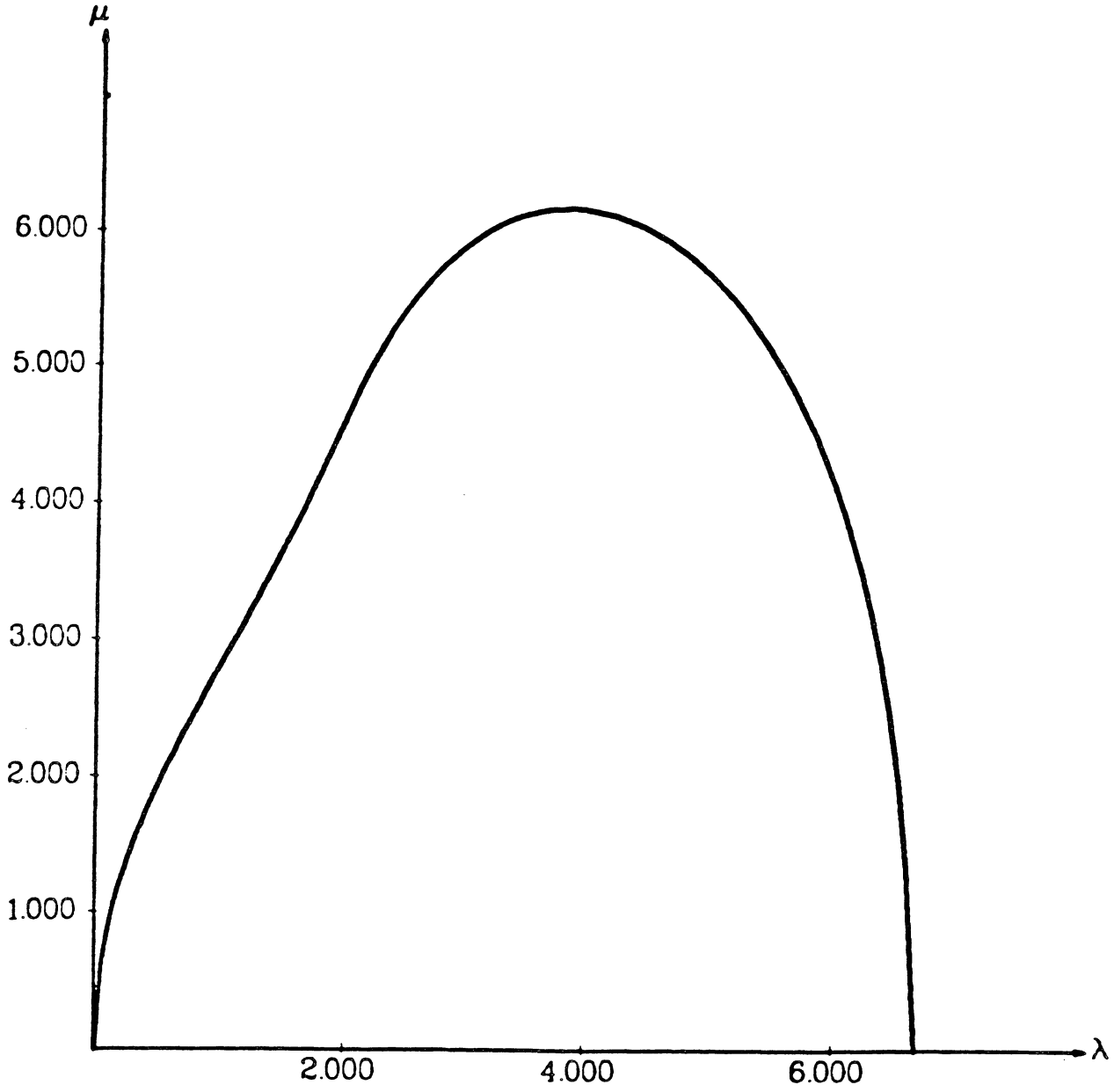

Figure 10a. Phase plane plot for straight line. 100 Interpolation points, velocity increment 0.025.

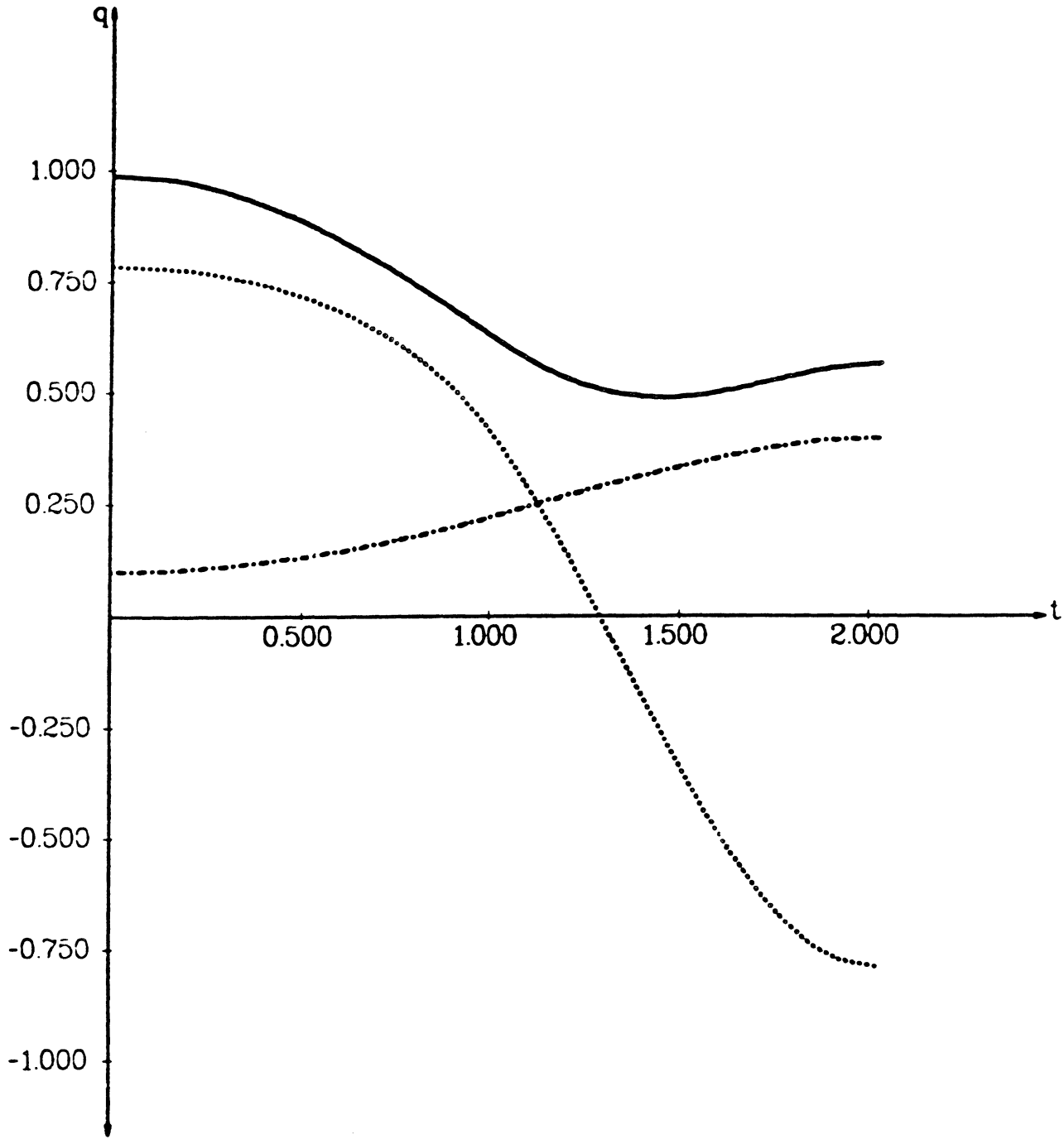

Figure 10b. Position vs. time for straight line. 100 Interpolation points, velocity increment 0.025.

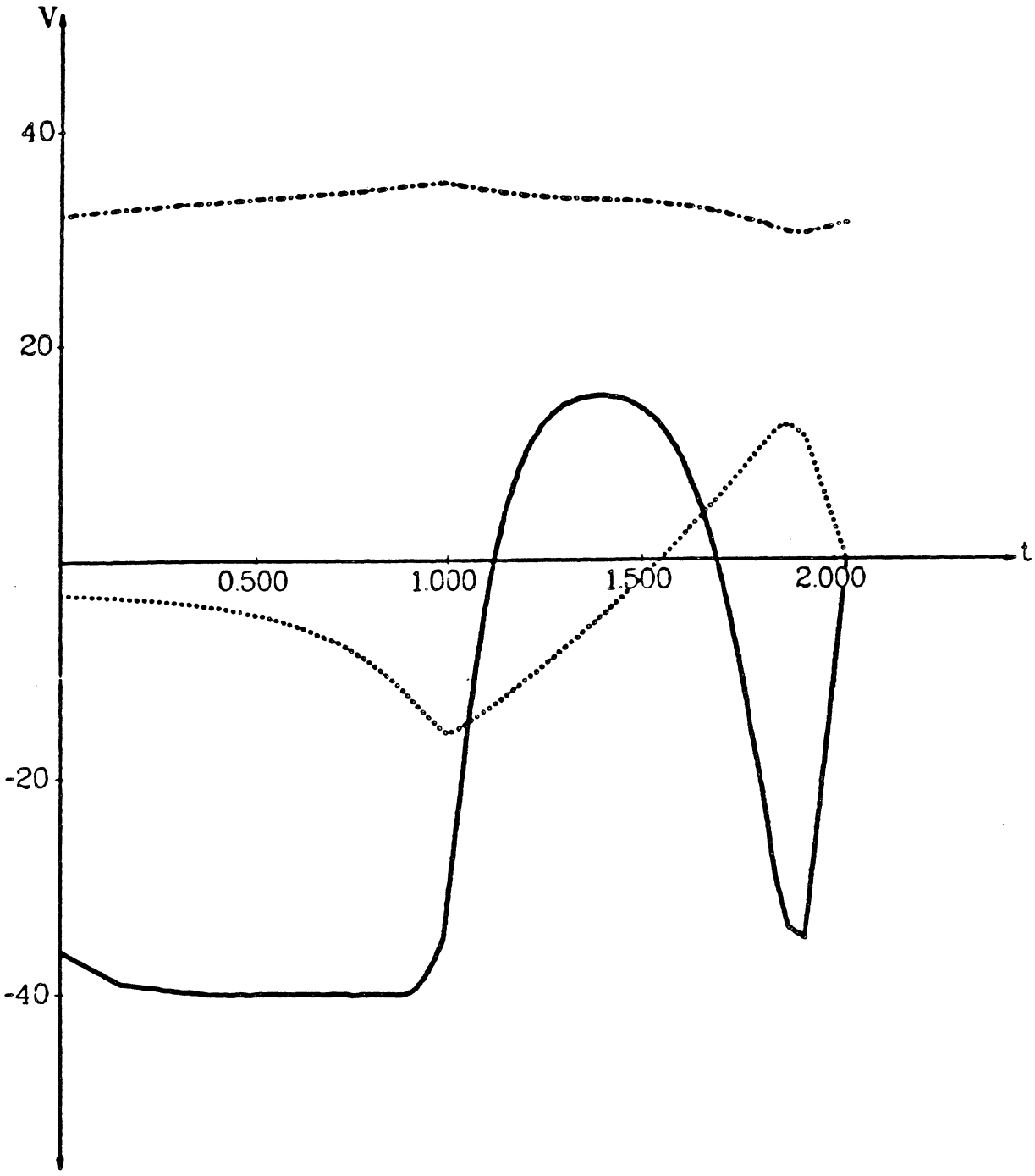

Figure 10c. Motor voltage vs. time for straight line. 100 Interpolation points, velocity increment 0.025.