Incorporating Body Dynamics Into Sensor-Based Motion Planning: The Maximum Turn Strategy

Andrei M. Shkel and Vladimir J. Lumelsky

***Abstract*—Most of today's approaches to sensor-based motion planning focus on kinematic and geometric issues and ignore the system dynamics. Those few that address dynamics do so in a two-stage fashion by considering one issue at the time. This work attempts to incorporate control of body dynamics into the sensor-based motion planning process. A point mobile robot is assumed to operate in a planar environment with unknown arbitrary stationary obstacles. Based on its current velocity and sensory data about the surrounding obstacles, the robot plans its motion to locally maximize the turning angle toward the current intermediate target. An optimal braking procedure takes care of sudden potential collisions by guaranteeing a safe emergency stopping path. Given the constraints on robot's dynamics, sensing, and control means, conditions are formulated for generating trajectories that guarantee convergence and the robot's safety at all times. The approach calls for continuous computation and is fast enough for real-time implementation. Simulated examples demonstrate its performance.**

***Index Terms*— Control, dynamics, obstacle avoidance, sensor-based motion planning**

I. Introduction

When planning motion for a machine, such as a robot, operating in a complex environment, one additional issue to consider is the machine's dynamics. A piece of trajectory that makes sense from the standpoint of the destination and the surrounding objects may not be feasible because of the effects of inertia and control limitations. This work studies those effects, with the goal of designing provably correct sensor-based algorithms for operating in an uncertain environment. Since the uncertainty implies that the global optimality is not feasible, it is natural to attempt a local optimization. A salient feature of the strategy suggested here is that while maintaining a maximum velocity that allows collision-free motion among the surrounding obstacles, the robot will attempt to locally maximize the turning angle toward the required direction of motion.

We consider a mobile robot operating in two-dimensional (2-D) physical space filled with a locally finite number of unknown stationary obstacles of arbitrary shapes. Planning is done in small steps (say, 30 or 50 times per second), resulting in continuous motion. The robot is equipped with sensors, such as vision or range finders, which allow it to detect and measure distances to surrounding objects within its sensing range—equal to, say, 20 or 50 steps. Therefore, unless obstacles occlude one another, the robot can see them far enough ahead to plan appropriate actions.

In addition to the usual problems of "where to go" and how to guarantee convergence in view of incomplete information, the robot's mass and velocity bring about the dynamic component of planning. A step that is reasonable from the standpoint of reaching the target position—for example, a sharp turn—may not be physically realizable because of the robot's inertia. Given the lack of information about the surroundings, this translates into a safety issue—one needs a guarantee of a *stopping path* at any time in case a sudden obstacle makes it impossible to continue on the intended path.

This is not unlike the decisions a human jogger faces when going for a morning run in an urban neighborhood. The jogger's speed, mass, quality of vision, and speed of reaction to sudden changes (the quality of control) will all be tied into some relationship, affecting the real-time decision-making process. Note that sensing, local planning, global planning, and actual movement in this process take place simultaneously and continuously. Locally, when the object is first noticed, unless a right relationship is maintained between the jogger's mass, velocity, and the distance to the object, a collision may occur; for example, a bigger mass may dictate better (farther) sensing to maintain the same velocity. Globally, unless a "grand plan" is followed, convergence may be lost.

Although system dynamics and sensor-based motion control are tightly coupled, little attention has been paid to this connection in the literature. Most of the existing approaches deal solely with the system kinematics and geometry and ignore its dynamic properties. Consequently, they can be used only in applications where the effect of speeds and masses is negligible. One reason for the difficulty is that the methods of motion planning tend to rely on tools from geometry and topology, which are not easily connected to the tools common to control theory.

Motion planning algorithms usually adhere to one of two paradigms that differ in their assumptions about input information. In the first paradigm, which is called *motion planning with complete information* (or the *Piano Mover's problem*), one assumes full information and algebraic representation of objects; motion planning is a one-time, off-line operation. In the second paradigm, which is called *motion planning with incomplete information* (or *sensor-based planning*), objects can be of arbitrary shape, and input information is of local character, such as from a range finder or vision. By making use of sensor feedback, this paradigm fits the on-line character of control theory methods well. Both paradigms give rise to two types of strategies: those that consider only kinematic and geometric issues—for brevity, we call them *kinematic approaches*—and those that take into account the system dynamics (we call them *dynamic approaches*).

Dynamics and control constraints can be incorporated into the Piano Mover's paradigm, for example, by dividing planning into two stages—first, one finds a path that satisfies geometric constraints and then modifies it to fit the dynamics constraints [1], possibly in a time-optimal fashion [2], [3]. One of the first attempts to explicitly incorporate body dynamics into the planning process were made by O'Dunlaing for the one-dimensional (1-D) case [4] and by Canny *et al* [5] in their kinodynamic planning approach for the 2-D case. The latter technique, although it operates in the context of complete information, is somewhat akin to our approach below in that the velocity and acceleration components are assumed to be bounded with respect to a fixed (absolute) reference system.

A number of strategies make use of the notion of artificial potential fields. Although attempts have been made to design such systems based on real-time sensory data [6], these strategies usually require complete information and analytical representation of obstacles; motion control makes use of "repulsive forces" modeled via a potential field associated with obstacles and of "attractive forces" associated with the goal position (see, e.g., [7]). A typical convergence issue

Manuscript received October 30, 1995; revised October 1, 1996. This work was supported in part by the Department of Energy (Sandia Labs) under Grant 18-4379C and the National Oceanic and Atmospheric Administration under Grant NA46RG048. This paper was recommended for publication by Associate Editor M. Peshkin and Editor S. Salcudean upon evaluation of the reviewers' comments.

The authors are with the Department of Mechanical Engineering, University of Wisconsin, Madison, WI 53706 USA (email: shkel@robios.me.wisc.edu; lumelsky@engr.wisc.edu)

Publisher Item Identifier S 1042-296X(97)07798-7.

here is how to avoid possible local minima in the potential field. An interesting approach called active reflex control [8] attempts to combine the potential field method with the handling of the robot dynamics; the emphasis is on local collision avoidance and on filtering out infeasible control commands generated by a "kinematic" planner.

Within the paradigm with incomplete information (which this work adheres to), a variety of kinematic techniques originate in maze-searching strategies [9], [10]. When applicable, they are typically fast, can be used in real time, and guarantee convergence; obstacles may be of arbitrary shapes.

To design a provably correct *dynamic* algorithm for sensor-based motion planning, one needs a single control mechanism—separating it into stages is likely to destroy convergence. Convergence has two faces: Globally, one has to guarantee finding a path to the target if one exists; locally, one needs an assurance of collision avoidance in view of the robot inertia. The former can be borrowed from kinematic algorithms; the latter requires an explicit consideration of dynamics.

In terms of the planning strategy, note that in spite of sufficient knowledge about the obstacles within the sensing range, at a given step, it would not be wise to address the problem as one with complete information and attempt to compute the whole subpath in the sensing range. The main reason for that is that only the first step of the subpath is likely to be executed as new sensing data at the next step will in general require path adjustment. That is, computing this whole subpath would be largely computational waste. Besides, computing this subpath would require solving a rather difficult optimal motion control problem. Given our various constraints, such as a limited operating area and obstacles, the solution would likely be computationally expensive, even for our simple model.

As usually in maze-searching strategies (see, e.g., [9]), sensory data will be only used for planning the immediately following step(s). No explicit (partial or full) model of the environment will be built. From the start point S, the algorithm will work its way toward the target point T sequentially in small steps, each of which i) lies on a globally convergent path and ii) satisfies the robot dynamics constraints.

The general strategy is as follows: At its current position C_i, the robot will identify a visible intermediate target point T_i that is guaranteed to lie on a convergent path and is far enough from the robot—normally, at the sensing range boundary. Since the direction toward T_i may differ from the current velocity vector $\mathbf{V}_i$, moving toward T_i may require a sharp turn, which may or may not be possible due to the system dynamics. If the angle between $\mathbf{V}_i$ and the direction toward T_i is larger than the maximum turn the robot can make in one step, the robot will attempt a fast smooth maneuver by turning at the maximum rate until the directions align, hence the name *maximum turn strategy*. (A case of time-optimal local optimization for the same task can be found in [11]). Once a step is executed, new sensing data appear, a new point T_{i+1} is sought, and so on. That is, the actual path and the path that contains points T_i are different paths—with the new sensory data at the next step, the robot may or may not actually pass through the point T_i.

The fact that no information is available beyond the sensing range dictates caution. First, to guarantee safety, the whole stopping path must lie inside the sensing range. Since some of this area may be occupied or occluded by obstacles, the stopping path must lie in its visible part. In addition, since the intermediate target T_i is chosen as the farthest point based on the information available, the robot needs a guarantee of stopping at T_i, even if it does not intend to do so. That is, each step is to be planned as the first step of a trajectory that, given the current position, velocity, and control constraints, would bring the robot to a halt at T_i. Within one step, the time to acquire sensory data and to calculate necessary controls must fit into the step cycle.

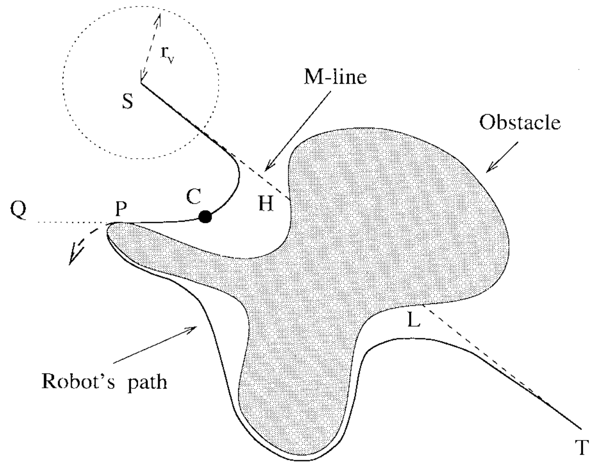


Fig. 1. Example of a conflict between the performance of a kinematic algorithm (the solid line path) and the effects of dynamics (the dotted piece of trajectory at P).

Below, the model, terminology used, and the problem statement are introduced in Section II, followed by a sketch of the suggested approach in Section III. After analysis of the system dynamics in Sections IV and V, the algorithm proper is presented, and its convergence properties are discussed in Section VI. A simulated example of the algorithm performance is given in Section VII.

II. The Problem Statement

The robot operates in a plane; the scene may include a locally finite number of static obstacles. Each obstacle is bounded by a simple closed curve of arbitrary shape and of finite length, such that a straight line will cross it only in a finite number of points. Obstacles do not touch each other (if they do, they are considered to be one obstacle). The total number of obstacles need not be finite.

The robot's sensors provide it with information about its surroundings within the *sensing range*, which is a disc of radius r_v ("radius of vision") centered at its current location C_i. Namely, it can assess the distance to the nearest obstacle in any direction within the sensing range. At moment t_i, the robot's input information includes its current velocity vector $\mathbf{V}_i$, coordinates of C_i and of the target point T, and possibly few other points of interest, such as an intermediate target T_i. The task is to move, collision-free, from point S (start) to point T (target); see Fig. 1.

The robot is a *point mass* of mass m. The motion control means include two components of the acceleration vector $\mathbf{u} = \frac{f}{m} = (p, q)$, where f is the force applied. Although the units of (p, q) are those of acceleration, by normalizing to $m = 1$, we can refer to p and q as control forces, each within its fixed range $|p| \leq p_{max}$, $|q| \leq q_{max}$; p and q are the only external forces acting on the system. Force p controls forward (or backward when braking) motion; its positive direction coincides with the velocity vector $\mathbf{V}$. Force q is perpendicular to p, forming a right pair of vectors, and is equivalent to the steering control (rotation of vector $\mathbf{V}$); see Fig. 2. Constraints on p and q imply a constraint on the path curvature. The point mass assumption implies that the robot's rotation with respect to its "center of mass" has no effect on the system dynamics. There is no friction; for example, values $p = q = 0$ and $\mathbf{V} \neq 0$ will result in a straight line constant velocity motion.[1]

Robot motion is controlled in *steps* $i = 0, 1, 2, \ldots$ Each step takes time $\delta t = t_{i+1} - t_i = const$; its length depends on the robot's

[1] If needed, other external forces and constraints can be handled within this model, using, for example, the technique described in [12].

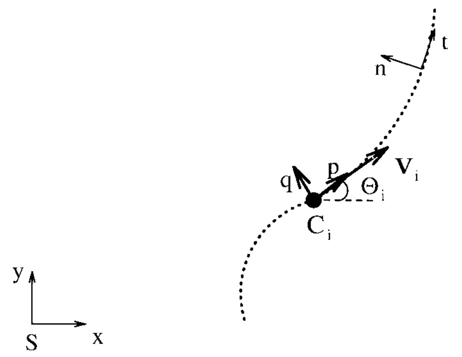


Fig. 2. Path coordinate frame $(\mathbf{t}, \mathbf{n})$ is used in the analysis of dynamic effects of robot motion. The world frame $(\mathbf{x}, \mathbf{y})$, with its origin at the start point S, is used in the obstacle detection and path planning analysis.

velocity within the step. Steps i and $i+1$ start at times t_i and t_{i+1}, respectively; $C_0 = S$. While moving toward location C_{i+1}, the robot computes necessary controls for step $i+1$ using the current sensory data and executes them at C_{i+1}. The finite time necessary within one step for acquiring sensory data, calculating the controls, and executing the step must fit into the step cycle (see details in [13]). We define two coordinate systems (follow Fig. 2):

- the *world coordinate frame* $(\mathbf{x}, \mathbf{y})$ fixed at point S;
- the *path coordinate frame* $(\mathbf{t}, \mathbf{n})$, which describes the motion of point mass at any moment $\tau \in [t_i, t_{i+1})$ within step i. Its origin is attached to the robot; axis $\mathbf{t}$ is aligned with the current velocity vector $\mathbf{V}$; axis $\mathbf{n}$ is normal to $\mathbf{t}$, i.e., when $\mathbf{V} = 0$, the frame is undefined. (One may note that together with axis $\mathbf{b} = \mathbf{t} \times \mathbf{n}$, the triple $(\mathbf{t}, \mathbf{n}, \mathbf{b})$ forms the known *Frenet trihedron*, with the plane of $\mathbf{t}$ and $\mathbf{n}$ being the *osculating plane* [14]).

III. The Maximum Turn Strategy

Define *M-line* (*Main line*) as the straight line segment (ST); see Fig. 1; this is the robot's desired path. When, while moving along the M-line, the robot senses an obstacle on its way, this point on the obstacle boundary is called a *hit point* H. The corresponding M-line point "on the other side" of the obstacle is a *leave point* L.

The planning procedure will be executed at each step of the robot's path. As said, any provable maze-searching algorithm can be used for the kinematic part, as long as it allows distant sensing. For specificity, we use here the VisBug algorithm [9], which alternates between these two steps (see Fig. 1).

1) Walk from point S toward point T along the M-line until detect an obstacle crossing the M-line.
2) Using sensing data, define the farthest visible *intermediate target* T_i on the obstacle boundary and on a convergent path; make a step toward T_i; iterate Step 2 until M-line is detected; go to Step 1.

In Fig. 1, note that under the VisBug procedure, while trying to pass the obstacle from the left, at point P, the robot makes a sharp turn. Such motion is not possible in a system with dynamics. To this, a control procedure for handling dynamics is added. At times, because of inertia or occluding obstacles, the current intermediate target T_i may go out of the robot's sight. In such cases, the robot will be designating *temporary intermediate targets* and use them until it can spot point T_i again. The actual algorithm also includes other mechanisms, such as a finite-time target reachability test and local path optimization.

Safety Considerations: Dynamics affects safety. Given the uncertainty beyond the distance r_v from the robot, a guaranteed *stopping path* is the only way to assure collision-free motion. Unless this "last resort" path is available, new obstacles may appear in the sensing range at the next step, and a collision may occur. A stopping path implies a safe direction of motion and a safe velocity value. We choose the stopping path to be a straight line segment along the step's velocity vector. A candidate step is "approved" only if its direction provides the stopping path. In this sense, the overall planning procedure is based on a one-step analysis.[2] The procedure for a detour around a sudden obstacle operates in a similar way.

For the continuous case, allowing a straight line stopping path with the stop point at the sensing range boundary implies the following relationship between the velocity $\mathbf{V}$, mass m, and controls $\mathbf{u} = (p, q)$.

$$V \leq \sqrt{2pd} \tag{1}$$

where d is the distance from the current position to the stop point. For example, an increase in the radius of vision r_v allows one to raise the maximum velocity by the virtue of providing more information farther along the path. Some ramifications of discrete control on this relationship are analyzed in Section IV.

Convergence: Because of dynamics, the convergence mechanism borrowed from a kinematic algorithm—here VisBug [9]—needs some modification. VisBug assumes that the intermediate target point is either on the boundary of the obstacle or on the M-line and is visible. However, the robot's inertia may cause it to move so that the intermediate target T_i will become invisible either because it goes outside the sensing range r_v (as after point P; see Fig. 1) or due to occluding obstacles (as in Fig. 6), with the danger that the robot may lose it and the path convergence with it. One possible solution is to keep the velocity low enough to avoid such overshoots—a high price in efficiency to pay. The solution chosen is to keep the velocity high and, if the intermediate target T_i does go out of sight, modify the motion locally until T_i is found again (Section VI).

IV. Velocity Constraints. Minimum Time Braking

By substituting p_{max} for p and r_v for d into (1), obtain the maximum velocity V_{max}. Since the maximum distance for which information is available is r_v, an emergency stop should be planned for that distance. We show that moving with the maximum speed (certainly a desired feature) actually guarantees a minimum-time stop at the boundary. The suggested braking procedure, developed in Section IV-C, makes use of an optimization scheme that is sketched briefly in Section IV-B.

A. Velocity Constraints

It is easy to see (follow an example in Fig. 3) that in order to guarantee a safe stopping path under discrete control, the maximum velocity must be less than V_{max}. This velocity, which is called *permitted maximum velocity* V_{pmax}, can be found from the following condition: If $V = V_{pmax}$ at point C_2 (and, thus, at C_1), we can guarantee the stop at the sensing range boundary (point B_1; see Fig. 3). Recall that velocity V is generated at C_1 by the control force p. Let $|C_1C_2| = \delta x$; then

$$\delta x = V_{pmax} \cdot \delta t$$
$$V_{B_1} = V_{pmax} - p_{max} t.$$

[2]While a deeper, multistep analysis could occasionally produce locally shorter paths, it would not add in safety and is not likely to justify the steep rise in computational expenses.

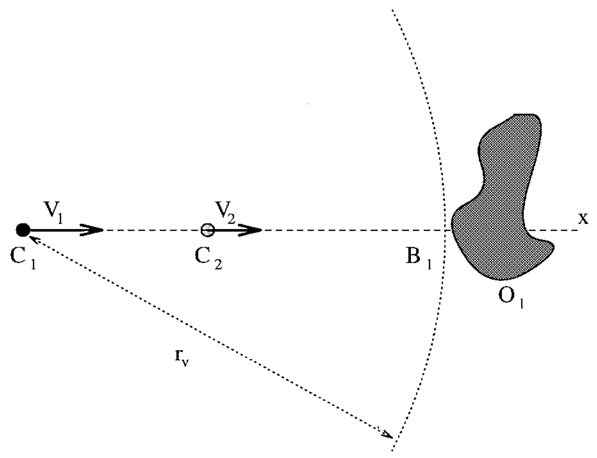


Fig. 3. With sensing radius r_v, obstacle O_1 is not visible from point C_1. Because of the discrete control, velocity V_1 commanded at C_1 will be constant during the step interval (C_1, C_2). Therefore, if $V_1 = V_{max}$ at C_1, then $V_2 = V_{max}$, and the robot will not be able to stop at B_1, causing collision with O_1. The permitted velocity, thus, must be $V_{pmax} < V_{max}$.

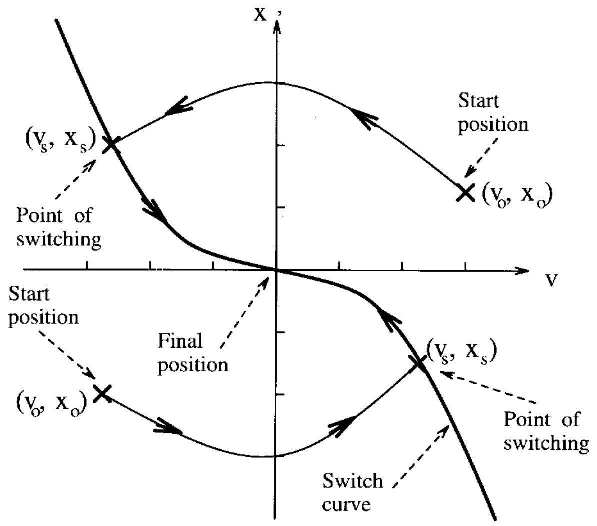


Fig. 4. Depending on whether the initial position (V_0, x_0) in the phase space (V, x) is above or below the switch curves, there are two cases to consider. The optimal solution corresponds to moving first from (V_0, x_0) to the point of switching (V_s, x_s) and then along the switch line to the origin.

Since we require $V_{B_1} = 0$, then $t = \frac{V_{pmax}}{p_{max}}$. For the segment $|C_2 B_1| = r_v - \delta x$, we have

$$r_v - \delta x = V_{pmax} \cdot t - \frac{p_{max} t^2}{2}.$$

From these equations, the expression for the maximum permitted velocity V_{pmax} can be obtained,

$$V_{pmax} = \sqrt{p_{max}^2 \delta t^2 + 2 p_{max} r_v} - p_{max} \delta t$$

As expected, $V_{pmax} < V_{max}$ and converges to V_{max} with $\delta t \to 0$.

B. Optimal Straight Line Motion

The following sketch of the optimization scheme is used below in the development of the braking procedure; for details, refer to [11]. Consider a dynamic system described by a second-order differential equation $\ddot{x} = p(t)$, where $\|p(t)\| \le p_{max}$, and $p(t)$ is a scalar control function. Assume that the system moves along a straight line. By introducing state variables x and V, the system equations can be rewritten as $\dot{x} = V$ and $\dot{V} = p(t)$; it is convenient to analyze the system behavior in the *phase space* (V, x).

The goal of control is to move the system from its initial position $(x(t_0), V(t_0))$ to the final position $(x(t_f), V(t_f))$. For convenience, choose $x(t_f) = 0$. We are interested in an optimal control strategy that would perform this motion in minimum time t_f, arriving at $x(t_f)$ with zero velocity $V(t_f) = 0$. This optimal solution can be obtained in closed form; it depends on the above/below relation of the initial position with respect to two parabolas that define the switch curves in the phase plane (V, x)

$$x = -\frac{V^2}{2p_{max}}, \quad V \ge 0 \tag{2}$$

$$x = \frac{V^2}{2p_{max}}, \quad V \le 0. \tag{3}$$

This simple result in optimal control (see. e.g, [15]), which is summarized in the following control law, is used in the next section to develop the braking procedure for the emergency stop; the procedure guarantees the robot's safety while allowing it to cruise with the maximum velocity (follow Fig. 4).

Control Law: If, in the phase space, the initial position (V_0, x_0) is above the switch curve (2), move first along the parabola defined by control $\hat{p} = -p_{max}$ toward curve (2) and then with control $\hat{p} = p_{max}$ along the curve to the origin. If point (V_0, x_0) is below the switch curve, first move with control $\hat{p} = p_{max}$ toward the switch curve (3) and then with control $\hat{p} = -p_{max}$ along the curve to the origin.

C. The Braking Procedure

We now turn to the calculation of time necessary for stopping when moving along the stopping path. It follows from the argument above that if at the moment when the robot decides to stop its velocity is $V = V_{pmax}$, then it will need to apply maximum braking all the way until the stop. This will define uniquely the time to stop. However, if $V < V_{pmax}$, then there is a variety of braking strategies and, hence, of different times to stop.

Consider again the example in Fig. 3; assume that at point C_2, $V_2 < V_{pmax}$. What is the *optimal braking strategy*, the one that guarantees safety while bringing the robot in minimum time to a stop at the boundary of the sensing range? (While this strategy is not necessarily what one would want to implement, it defines the limit velocity profile the robot can maintain for safe braking.) The answer is given by solving an optimization problem for a single degree-of-freedom system. It follows from the control law that the optimal solution corresponds to at most two curves I and II in the phase space (V, x) [see Fig. 5(a)] and to at most one control switch, from $\hat{p} = p_{max}$ on line I to $\hat{p} = -p_{max}$ on line II, given by (2) and (3). For example, if braking starts with the initial values $x = -r_v$ and $0 \le V_0 < V_{max}$, the system will first move, with control $\hat{p} = p_{max}$, along the parabola I to the parabola II [see Fig. 5(a)]

$$x(V) = \frac{V^2 - V_0^2}{2p_{max}} - r$$

and then, with control $\hat{p} = -p_{max}$, toward the origin along the parabola II

$$x(V) = \frac{V^2}{2p_{max}}.$$

The optimal time t_b of braking is a function of the initial velocity V_0, radius of vision r_v, and the control limit p_{max},

$$t_b(V_0) = \frac{\sqrt{2V_0^2 + 4p_{max} r_v} - V_0}{p_{max}}. \tag{4}$$

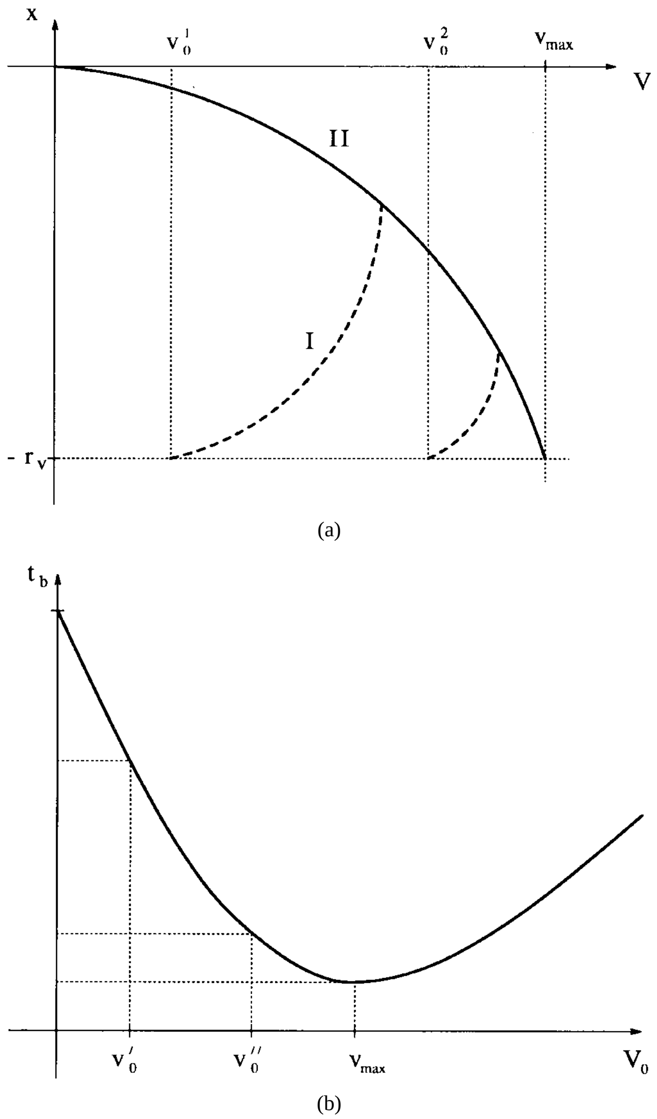


Fig. 5. (a) Optimal braking strategy requires at most one switch of control. (b) Corresponding time-velocity relation.

Function $t_b(V_0)$ has a minimum at $V_0 = V_{max} = \sqrt{2p_{max}r_v}$, which is exactly the upper bound on the velocity given by (1); it is decreasing on the interval $V_0 \in [0, V_{max}]$ and increasing when $V_0 > V_{max}$; see Fig. 5(b). For the interval $V_0 \in [0, V_{max}]$, which is of interest to us, the above analysis leads to a somewhat counter-intuitive conclusion.

Proposition 1: For the initial velocity V_0 in the range $V_0 \in [0, V_{max}]$, the time necessary for stopping at the boundary of the sensing range is a monotonically decreasing function of V_0, with its minimum at $V_0 = V_{max}$.

Notice that this result (see also Fig. 5) leaves a comfortable margin of safety. Even if, at the moment when the robot sees an obstacle, it moves with the maximum velocity, it can still stop safely before the obstacle. If the robot's velocity is below the maximum, it has different options for braking, including even one of speeding up before actual braking. Assume, for example, we want to stop in minimum time at the sensing range boundary [the origin in Fig. 5(a)]; consider two initial positions: i) $x = -r_v$, $V = V_0^1$, and ii) $x = -r_v, V = V_0^2; V_0^2 > V_0^1$. Then, according to Proposition 1, in case i), this time is bigger than in case ii). Note that because of the discrete control, it is the permitted maximum velocity V_{pmax} that is to be substituted into (4) to obtain the minimum time.

V. Dynamics and Collision Avoidance

The following analysis consists of two parts. First, the control constraints are incorporated into the mobile robot model, and the transformation from the moving path coordinate frame to the world frame is developed (see Section II). Then, the maximum turn strategy is presented, an incremental decision-making mechanism that determines the forces p and q at each step.

A. Transformation from Path Frame to World Frame

The remainder of this section refers to the time interval $[t_i, t_{i+1})$; therefore, index i can be dropped. Let $(x, y) \in \mathcal{R}^2$ be the robot's position in the world frame, and let θ be the (slope) angle between the velocity vector $\mathbf{V} = (V_x, V_y) = (\dot{x}, \dot{y})$ and x-axis of the world frame (see Fig. 2). The planning process involves computation of the controls $\mathbf{u} = (p, q)$, which for every step defines the velocity vector and eventually the path $\mathbf{x} = (x, y)$ as a function of time. The normalized equations of motion are

$$\ddot{x} = p\cos\theta - q\sin\theta$$
$$\ddot{y} = p\sin\theta + q\cos\theta.$$

The angle θ between vector $\mathbf{V}$ and the x-axis of the world frame is found as

$$\theta = \begin{cases} \arctan(\frac{V_y}{V_x}), & V_x \geq 0 \\ \arctan(\frac{V_y}{V_x}) + \pi, & V_x < 0. \end{cases}$$

To find the transformation from the path frame to the world frame $(\mathbf{x}, \mathbf{y})$, present the velocity in the path frame as $\mathbf{V} = V\mathbf{t}$. Angle θ is defined as the angle between $\mathbf{t}$ and the positive direction of the x axis. Given that the control forces p and q act along the $\mathbf{t}$ and $\mathbf{n}$ directions, respectively, the equations of motion with respect to the path frame are

$$\dot{V} = p,$$
$$\dot{\theta} = q/V.$$

Since the control forces are constant over time interval $[t_i, t_{i+1})$, within this interval, the solution for $V(t)$ and $\theta(t)$ becomes

$$V(t) = V_0 + pt,$$
$$\theta(t) = \theta_0 + \frac{q\log(1 + \frac{tp}{V_i})}{p} \tag{5}$$

where θ_0 and V_0 are constants of integration and are equal to the values of $\theta(t_i)$ and $V(t_i)$, respectively. By parameterizing the path by the value and direction of the velocity vector, the path can be mapped onto the world frame using the vector integral equation

$$\mathbf{r}(t) = \int_{t_i}^{t_{i+1}} \mathbf{V} \cdot dt. \tag{6}$$

Here, $\mathbf{r}(t) = (x(t), y(t))$, and $\mathbf{V} = (V \cdot \cos(\theta), V \cdot \sin(\theta))$ are the projections of vector $\mathbf{V}$ onto the world frame $(\mathbf{x}, \mathbf{y})$. After integrating (6), we obtain the set of solutions in the form

$$x(t) = \frac{2p\cos\theta(t) + q\sin\theta(t)}{4p^2 + q^2}V^2(t) + A$$
$$y(t) = -\frac{q\cos\theta(t) - 2p\sin\theta(t)}{4p^2 + q^2}V^2(t) + B \tag{7}$$

where terms A and B are

$$A = x_0 - \frac{V_0{}^2\,(2\,p\,\cos(\theta_0) + q\,\sin(\theta_0))}{4\,p^2 + q^2}$$
$$B = y_0 + \frac{V_0{}^2\,(q\,\cos(\theta_0) - 2\,p\,\sin(\theta_0))}{4\,p^2 + q^2}.$$

Equations (7) are directly defined by the control variables p and q; $V(t)$ and $\theta(t)$ therein are given by (5).

In general, (7) describes a spiral curve. Note two special cases. When $p \neq 0, q = 0$, (7) describes a straight line motion along the vector of velocity; when $p = 0, q \neq 0$, (7) produces a circle of radius $V_0^2/|q|$ centered at the point (A, B).

B. Selection of Control Forces

We now turn to the control law that guides the selection of forces p and q at each step i for the time interval $[t_i, t_{i+1})$. To assure a reasonably fast convergence to the intermediate target T_i, those forces are chosen such as to align, as fast as possible, the direction of the robot's motion with that toward T_i. First, find a solution among the controls (p, q) such that

$$(p,q) \in \{(p,q) : p \in [-p_{max},\, +p_{max}], q = \pm q_{max}\} \tag{8}$$

where $q = +q_{max}$ if the intermediate target T_i lies in the left semiplane and $q = -q_{max}$ if it lies in the right semiplane, with respect to the vector of velocity. That is, force p is chosen to keep the maximum velocity allowed by the surrounding obstacles. To this end, a discrete set of values p is tried until a step that guarantees a collision-free stopping path is found. At a minimum, the set should include values $-p_{max}, 0$, and $+p_{max}$; the more values that are tried, the closer the resulting velocity is to the maximum sought. Force q is chosen on the boundary to produce a maximum turn in the appropriate direction. On the other hand, if because of obstacles no adequate controls in the range (8) can be chosen, this means that maximum braking should be applied. Then, the controls are chosen from the set

$$(p,q) \in \{(p,q) : p = -p_{max}, q \in (\pm q_{max}, 0]\} \tag{9}$$

where q is found from a discrete set similar to p in (8). Note that the sets (8) and (9) always include at least one safe solution. By the algorithm's design, the straight line motion with maximum braking $(p, q) = (-p_{max}, 0)$ is always collision free (for more details, see [13]).

VI. The Algorithm

The algorithm consists of three procedures: The *Main Body* procedure defines the motion toward the intermediate target T_i within the time interval $[t_i, t_{i+1})$, the *Define Next Step* procedure chooses forces p and q, and the *Find Lost Target* procedure handles the case when the intermediate target goes out of the robot's sight. A procedure called *Compute* T_i, from the VisBug algorithm [9], which computes the next intermediate target T_{i+1} and includes a test for target reachability, is also used. Vector $\mathbf{V}_i$ is the current vector of velocity, and T is the robot's target. The term "safe motion" refers to the mechanism for determining the next robot's position C_{i+1} with the stopping path guarantee (see Section III).

Main Body: The procedure is executed at each time interval $[t_i, t_{i+1})$ and makes use of two procedures: *Define Next Step* and *Find Lost Target*:

- M1: Move in the direction specified by *Define Next Step* while executing *Compute* T_i. If T_i is visible, do: if $C_i = T$, the procedure stops; else, if T is unreachable, the procedure stops; else, if $C_i = T_i$, go to M2. Otherwise, use *Find Lost Target* to make T_i visible. Iterate M1.
- M2: Make a step along vector $\mathbf{V}_i$ while executing *Compute* T_i: if $C_i = T$, the procedure stops; else, if the target is unreachable, the procedure stops; else, if $C_i \neq T_i$, go to M1.

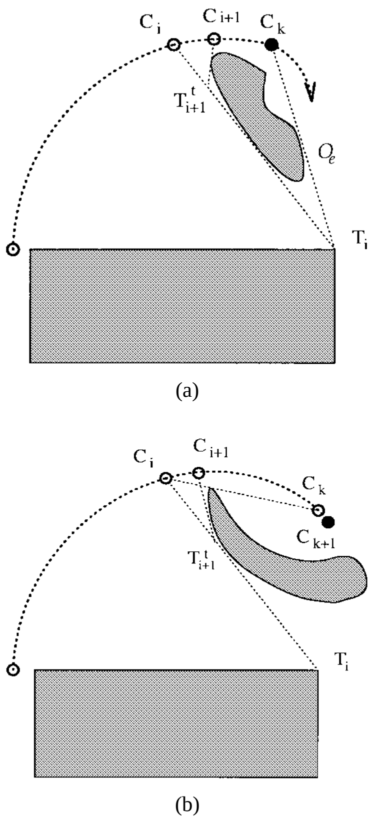
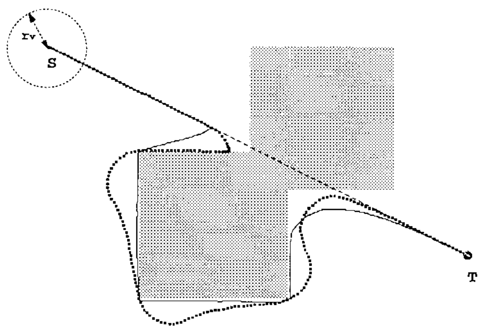


Fig. 6. Because of inertia, the robot here temporarily "loses" the intermediate target T_i. In (a), it keeps moving around the obstacle until it spots T_i and then continues toward it. In (b), when the whole segment (C_i, T_i) becomes invisible at point C_k, the robot stops at point C_{k+1}, returns back to C_i, and then moves toward T_i along the line (C_i, T_i).

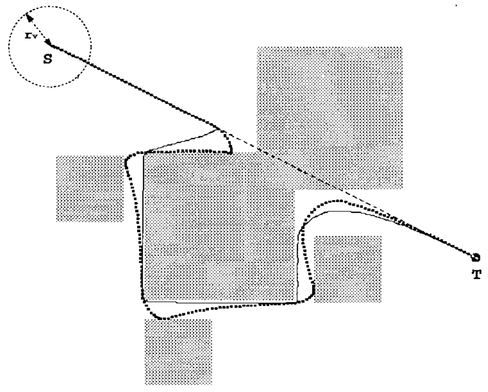
Define Next Step: This procedure covers all possible cases of generation of a single motion step: Part D1 corresponds to motion along M-line; D2 is a simple turn when the directions of vectors $\mathbf{V}_i$ and (C_i, T_i) can be aligned in one step; D3 occurs when the turn requires multiple steps and can be done with the maximum speed; D4 happens when turning must be accompanied by braking:

- D1: If vector $\mathbf{V}_i$ coincides with the direction toward T_i, do: if $T_i = T$, make a step toward T; else, make a step toward T_i.
- D2: If vector $\mathbf{V}_i$ does not coincide with the direction toward T_i, do: if the directions of $\mathbf{V}_{i+1}$ and (C_i, T_i) can be aligned within one step, choose this step. Else, go to D3.
- D3: If a step with the maximum turn toward T_i and with maximum velocity is safe, choose it. Else, go to D4.
- D4: If a step with the maximum turn toward T_i and some braking is possible, choose it. Else, choose a step along $\mathbf{V}_i$ with maximum braking $p = -p_{max}, q = 0$.

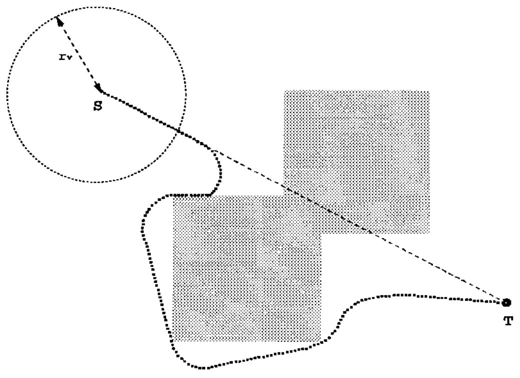
Find Lost Target: This procedure is executed when T_i becomes invisible. The last position C_i, where T_i was visible, is kept in the memory until T_i becomes visible again. A simple though inefficient strategy could be to immediately initiate a stopping path, then come back to C_i, and then move directly toward T_i. Instead, the procedure operates as follows: If the robot loses T_i, it keeps moving ahead while defining temporary intermediate targets on the visible part of the line segment (C_i, T_i) and continuing to look for T_i. If it sees T_i, the procedure terminates, the control returns to the *Main Body*, and the robot moves directly toward T_i; see Fig. 6(a). Otherwise, if the whole segment (C_i, T_i) becomes invisible, the robot brakes to a stop and returns to C_i, the procedure terminates, etc.; see Fig. 6(b). Together,



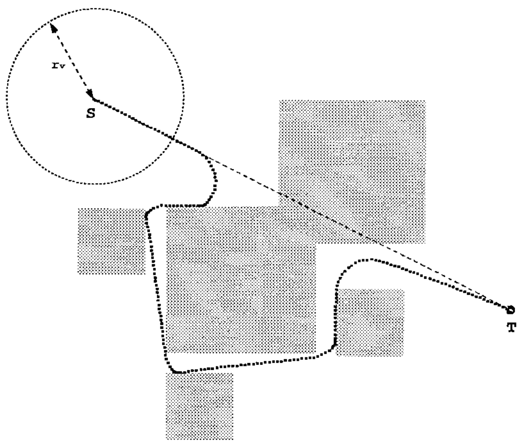
(a)



(b)



(c)



(d)

Fig. 7. In the examples, one path (indicated by the thin line) is produced by the kinematic algorithm, and the other path (a thicker line) — by the maximum turn strategy, with dynamics taken into consideration; The radius of vision r_v is also shown.

these two pieces assure that T_i will not be lost. The procedure is as follows.

- F1: If segment (C_i, T_i) is visible, define on it a temporary intermediate target T_i^t and move toward it while looking for T_i. If the current position is at T, exit; else, if C_i lies in the segment (C_i, T_i), exit. Else, go to F2.
- F2: If segment (C_i, T_i) is invisible, initiate a stopping path, and then go back to C_i; exit.

Convergence: To prove convergence, we need to show that

i) at every step of the path, the algorithm guarantees collision-free motion;
ii) the set of intermediate targets T_i is guaranteed to lie on the convergent path;
iii) the overall motion planning strategy assures that the current intermediate target will not be lost.

Together, ii) and iii) assure that a path to the target position T will be found if one exists. Condition i) can be shown by induction; condition ii) is provided by the VisBug mechanism [9], which also includes a mechanism for inferring the nonreachability of T, if true; condition iii) is satisfied by the procedure *Find Lost Target* of the Maximum Turn Strategy. The following statements hold.

Proposition 2: Under the Maximum Turn Strategy algorithm, assuming zero velocity at the start position S, $\mathbf{V}_S = 0$, at each step of the path, there exists at least one stopping path.

Indeed, according to our approach, the stopping path is a straight line segment. Choosing the next step to guarantee the existence of a stopping path implies two requirements: a safe direction of motion and a velocity value that would allow a stop within the visible area. The latter is assured by the choice of the system parameters [see (1) and the safety conditions in Section III]. As to the existence of safe directions, we proceed by induction: We need to show that if a safe direction exists at the start point and at an arbitrary step i, then there is a safe direction at the step $(i+1)$.

Since at the start point S the velocity is zero $\mathbf{V}_S = 0$, then any direction of motion at S will be a safe direction; this gives the basis of induction. The induction proceeds as follows. Under the algorithm, a candidate step is accepted for execution only if its direction guarantees a safe stop for the robot if needed. Namely, at point C_i, step i is executed only if the resulting vector $\mathbf{V}_{i+1}$ at C_{i+1} will point in a safe direction. Therefore, at step $(i+1)$, at the least, direction V_{i+1} presents a safe stopping path.

Remark: Proposition 2 will hold for $\mathbf{V}_S \neq 0$ as well if the start point S is known to possess at least one stopping path originating at it.

Proposition 3: The Maximum Turn Strategy is convergent.

To see this, note that by the design of the kinematic algorithm [9], each intermediate target T_i lies on a convergent path. In addition, T_i is visible at the moment when it is generated. That is, the only way the robot can get lost is if at the next steps C_{i+1}, point T_i becomes invisible due to the robot's inertia or an obstacle occlusion: This would make it impossible to generate the next intermediate target T_{i+1}, as required by the kinematic algorithm. However, if point T_i does become invisible, the procedure *Find Lost Target* is invoked, a set of temporary intermediate targets T_{i+1}^t are defined, and associated steps are executed until point T_i becomes visible again (see Fig. 6). The set T_{i+1}^t is finite since it must lie within the sensing range of radius r_v, and the algorithm chooses each T_{i+1}^t to guarantee a stopping path. Therefore, the robot always moves toward a point that lies on a path that is convergent to the target T.

VII. Examples

The following examples in Fig. 7(a)–(d) show the performance of the Maximum Turn Strategy in a simulated environment. The "dynamic" paths generated by the algorithm are shown in thicker lines. For comparison, paths produced under the same conditions by a kinematic (here, VisBug) algorithm are shown by a thin line. (Note that the polygonal shape of the obstacles in the examples is only for the convenience of generating the scene; the algorithms are immune to the obstacles' shape).

To understand the examples, consider a simplified version of the relationship that appears in Section IV-A, $V_{max} = \sqrt{2r_v p_{max}} = \sqrt{2r_v \cdot f_{max}/m}$. In the simulations, the robot's mass m and control force f_{max} are kept constant, e.g., an increase in sensing radius r_v would "raise" the velocity V_{max}. Radius r_v is the same in Fig. 7(a) and (b). A more complex scene [Fig. 7(b)] causes the robot to move more "cautiously," that is, slower; the path is tighter and closer to the obstacles. Accordingly, the time to complete the task is 221 units (steps) in Fig. 7(a) and 232 units in Fig. 7(b), whereas the length of paths is shorter in Fig. 7(b) than in Fig. 7(a).

Fig. 7(c) and (d) illustrate that better sensing (larger r_v) resulted in shorter time to complete the task; more crowded space resulted in longer time (though perhaps in shorter paths). The time to complete the task is 193 units in Fig. 7(c) and 209 units in Fig. 7(d).

Note that stops along the path (indicated by sharp turns in the "dynamic" path) can be caused by different reasons, e.g., in Fig. 7(a), the robot stops because its small sensing radius r_v is not sufficient to see the obstacle far enough to initiate a smooth turn.

References

[1] Z. Shiller and H. H. Lu, "Computation of path constrained time optimal motions along specified paths," *ASME J. Dyn. Syst., Meas. Contr.*, vol. 114, no. pp. 34–40, Mar. 1992.

[2] J. Bobrow, "Optimal robot path planning using the minimum-time criterion," *IEEE J. Robot. Automat.*, vol. 4, pp. 443–450, Aug. 1988.

[3] Z. Shiller and S. Dubowsky, "On computing the global time optimal motions of robotic manipulators in the presence of obstacles," *IEEE Trans. Robot. Automat.*, vol. 7, pp. 785–797, 1991.

[4] C. O'Dunlaing, "Motion planning with inertial constraints," *Algorithmica*, vol. 2, no. 4, 1987.

[5] J. Canny, A. Rege, and J. Reif, "An exact algorithm for kinodynamic planning in the plane," in *Proc. 6th Annu. Symp. Comput. Geometry*, Berkeley, CA, June 1990.

[6] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–99, 1986.

[7] C. De Medio and G. Oriolo, "Robot obstacle avoidance using vortex fields," in *Advances in Robot Kinematics*, S. Stifter and J. Lenarcic, Eds. New York: Springer-Verlag, 1991.

[8] T. Wikman, M. Branicky, and W. Newman, "Reflexive collision avoidance: A generalized approach," in *Proc. IEEE Int. Conf. Robot. Automat.*, Raleigh, NC, May 1993.

[9] V. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, pp. 1058–1069, 1990.

[10] A. Sankaranarayanan and M. Vidyasagar, "Path planning for moving a point object amidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm development," in *Proc. 29th IEEE Int. Conf. Decision Contr.*, Honolulu, HI, 1990.

[11] A. Shkel and V. Lumelsky, "The Jogger's problem: Control of dynamics in real-time motion planning," *Automatica*, vol. 33, pp. 1219–1233, July 1997.

[12] T. Fraichard and A. Scheuer, "Car-like robots and moving obstacles," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Diego, CA, May 1994.

[13] A. Shkel and V. Lumelsky, "The role of time constraints in the design of control for the Jogger's problem," in *Proc. IEEE Int. Conf. Decision Contr.*, New Orleans, LA, Dec. 1995.

[14] G. Korn and T. Korn, *Mathematical Handbook*. New York: McGraw-Hill, 1968.

[15] L. Hocking, *Optimal Control*. Oxford, U.K.: Clarendon, 1991.

Robust and Accurate Time-Optimal Path-Tracking Control for Robot Manipulators

Jon Kieffer, Aidan J. Cahill, and Matthew R. James

***Abstract*—The well-known algorithms for time-optimal trajectory planning are difficult to apply in practice because they rely on imperfect models of the robot dynamics, they take no account of controller dynamics, and they provide no connection to tracking accuracy. In this paper we propose two schemes for planning and implementing time-optimal control to enable robots under computed torque control to track paths to a prescribed tolerance. Experimental results confirm the theories, showing that both schemes track to a prescribed accuracy in a near time-optimal fashion. The second scheme is shown to perform better, achieving complete torque utilization nearly all of the time.**

***Index Terms*—Path tracking, robotics, time-optimal control.**

I. Introduction

Interest in the problem of controlling robots to track paths in minimum time is motivated by the desire to reduce cycle times in industrial applications such as laser cutting and high pressure water jet cutting. It may also be important for welding, glue dispensing, and spray painting operations, but only if speed is limited by the robot actuator constraints rather than by the cutting, gluing, welding, or painting process. In most path-tracking applications accuracy is vital, and it is reasonable to assume that designers and manufacturers will specify precision using an absolute tolerance on tracking error.

Well-known approaches to planning time-optimal trajectories for prescribed paths [14] are based on knowing the robot's governing dynamics and applying the path constraint to those dynamics to define a lower dimensional dynamic system. Typically, this reduces the dimension of the dynamics from 12, the joint positions and velocities to two, the path position, and velocity. This reduction alleviates the *curse of dimensionality in dynamic optimization*, making it practical to compute time-optimal solutions by phase plane shooting methods or by dynamic programming.

The resulting trajectories provide minimum time solutions for the considered dynamic systems, but their importance in real applications needs some clarification. The resulting *torque* trajectories cannot be applied directly to the real robot in an open-loop fashion because modeling errors will accumulate, leading to severe tracking errors. Thus a closed-loop tracking controller is needed.

The "time-optimal" *joint* trajectories provided by these well-known algorithms might be expected to provide a good reference trajectory for the closed-loop tracking controller, but we argue that this is unlikely for the following reasons.

1) The trajectories are planned without considering uncertainties in the dynamic model.

Manuscript received July 22, 1996; revised December 6, 1996. This work was supported in part by the Cooperative Research Centre for Robust and Adaptive Systems, by the Australian Government under the Cooperative Research Centres Program, and the Australian Research Council. This paper was recommended for publication by Associate Editor V. Kumar and Editor A. J. Koivo upon evaluation of the reviewers' comments.

J. Kieffer and M. R. James are with the Department of Engineering, Faculty of Engineering and Information Technology, the Australian National University, Canberra, ACT 0200, Australia (e-mail: Jon.Kieffer@anu.edu.au).

A. J. Cahill is with CAE Electronics, Silverwater, NSW 2128, Australia.

Publisher Item Identifier S 1042-296X(97)07800-2.