Alonzo Kelly
Bryan Nagy

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890, USA
alonzo@ri.cmu.edu
bnagy@rec.ri.cmu.edu

Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control

Abstract

There are many situations for which a feasible nonholonomic motion plan must be generated immediately based on real-time perceptual information. Parametric trajectory representations limit computation because they reduce the search space for solutions (at the cost of potentially introducing suboptimality). The use of any parametric trajectory model converts the optimal control formulation into an equivalent nonlinear programming problem. In this paper, curvature polynomials of arbitrary order are used as the assumed form of solution. Polynomials sacrifice little in terms of spanning the set of feasible controls while permitting an expression of the general solution to the system dynamics in terms of decoupled quadratures. These quadratures are then readily linearized to express the necessary conditions for optimality. Resulting trajectories are convenient to manipulate and execute in vehicle controllers and they can be computed with a straightforward numerical procedure in real time.

KEY WORDS—mobile robots, car-like robots, trajectory generation, curve generation, nonholonomic, clothoid, cornu spiral, optimal control

1. Introduction

Trajectory generation is a more difficult problem than it may at first appear to be. By contrast to manipulation, where the common inverse problem is that of inverting nonlinear kinematic equations, the common inverse problem for mobile robots is that of inverting nonlinear differential equations.

1.1. Notation

For a vehicle actuated in curvature and speed while moving in the plane, one description of its dynamics is the following four coupled, nonlinear equations:

$$\begin{aligned} \dot{x}(t) &= V(t)\cos\theta(t) \quad \dot{\theta}(t) = \kappa(t)V(t) \\ \dot{y}(t) &= V(t)\sin\theta(t) \quad \dot{\kappa}(t) = u(t). \end{aligned} \tag{1}$$

The vehicle state vector (also called a posture in this context) consists of the position coordinates (x, y), heading θ, and curvature κ:

$$\mathbf{x} = (x, y, \theta, \kappa)^{\mathrm{T}}. \tag{2}$$

The input or control vector consists of speed V and desired curvature u:

$$\mathbf{u} = (V, u)^{\mathrm{T}}. \tag{3}$$

It is straightforward to effect a change of variable from time to distance. Integrating the equations produces a canonical expression of the equations of odometric dead reckoning:

$$\begin{aligned} x(s) &= \int_0^s \cos\theta(s)\mathrm{d}s \qquad \theta(s) = \int_0^s \kappa(s)\mathrm{d}s \\ y(s) &= \int_0^s \sin\theta(s)\mathrm{d}s \qquad \kappa(s) = u(s). \end{aligned} \tag{4}$$

These equations are not a solution in the classical differential equation sense because the heading (a state) appears inside the integrals.

1.2. Problem Statement

The forward problem is that of determining the state space trajectory from the input functions. This problem is equivalent to dead reckoning and it can be solved by integrating the above equations numerically. It is not possible to compute the position without simultaneously computing the heading, because the above is not formally a solution.

In this paper we address the inverse problem. In the inverse problem, all or part of the state space trajectory **x** is

The International Journal of Robotics Research
Vol. 22, No. 7–8, July–August 2003, pp. 583-601,
©2003 Sage Publications

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

specified, and the associated control **u** (input function) must be computed. The question of whether such a **u** exists when boundary conditions are specified is one of classical controllability. When more than one such input exists, it becomes possible to think about optimizing some performance index (such as smoothness) and the problem becomes one of optimal control.

Unlike in the case of holonomic motion planning, obstacles are not required in order to make the problem of nonholonomic trajectory generation difficult. The terms trajectory generation, trajectory planning, and nonholonomic motion planning have been used historically for the problem of achieving goal postures while respecting dynamic and nonholonomic limitations on mobility. In many cases, quantities to be optimized are introduced and, less frequently, known obstacles are introduced to generate additional constraints on mobility and require higher degrees of search.

Much of the work to date has either expressed the problem strictly in terms of goal posture acquisition or assumed that the environment was known a priori. Yet, every time that an operating vehicle must react to its environment based on sensed information gathered while on the move, a nonholonomic motion plan must be generated in real time. Indeed, our work in applications points to a strong need for feasible motion plans to be generated virtually instantaneously in response to newly acquired environmental information. In this paper, we address the need to generate such reactive trajectories in real time.

1.3. Motivation

Real-time trajectory generation is motivated by applications of precision control. While computing trajectories is a complicated matter, there are many situations for which nothing less will solve the problem. Due to dynamics, limited curvature, and underactuation, a vehicle often has few options for how it travels over the space immediately in front of it. The key to achieving a relatively arbitrary posture is to think about doing so well before getting there, and to do so based on precise understanding of the above limitations.

One of the motivations for our work on this problem is the application of robot fork trucks handling pallets in factories, as illustrated in Figure 1. Pallets can only be picked up when addressed from a posture which places the fork tips at the fork holes with the right heading and with zero curvature. In our application, a vision system determines where the fork holes are, so the goal posture may not be known until limited space requires an aggressive maneuver to address the load correctly.

In the event that the fork holes are located after traveling past the point where a feasible capture motion exists, it still may be valuable to optimize the terminal posture error based on the fact that the holes are often much larger than the forks.

Obstacle avoidance also requires precise models of mobility. In Figure 2, for example, the space of constant curvature arc trajectories does not contain a solution to the problem

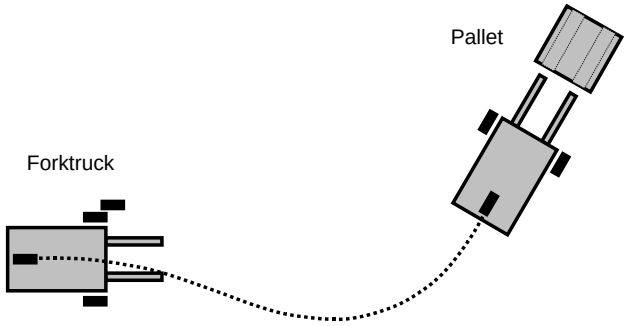

Fig. 1. Robot fork truck motivation. Based on the location of the load, a trajectory must be generated which ends precisely in front of, and aligned with, the fork holes. The curvature and speed must also be zero at the terminal point. Although the pallet is to the left of the truck, it must turn initially to the right to achieve the goal posture.

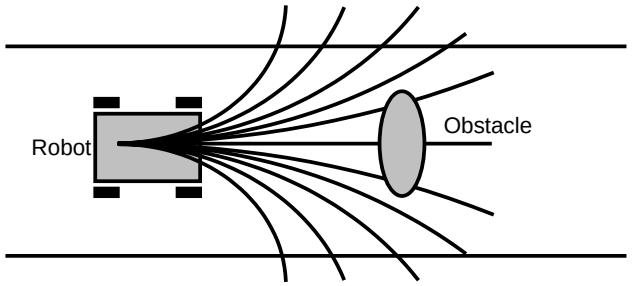

Fig. 2. Obstacle avoidance motivation. The robot must avoid the obstacle while staying on the road. Given the length of the arcs being evaluated (which reflects the stopping distance at this speed), there is no curvature which does not hit either the obstacle or the road edge. Yet, a compound curve easily avoids both.

of both staying on the road and avoiding the obstacle. However, the space of all feasible vehicle motions does contain a solution.

Simplistic approaches can quickly lead to unnecessary problems. In the fork truck example, simply steering toward the pallet is exactly the wrong thing to do. The only way to achieve the goal is to turn away from the pallet in order to lengthen the path enough to achieve the required heading change.

In summary, a mobile robot must precisely understand and exploit its own capacity to maneuver in order to function effectively in realistic applications. Several other applications for trajectory generation include the following.

- Coordinated control. By expressing curvature as a function of distance, it is straightforward to slave the steer-

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

ing wheel to the actual distance traveled in order to ensure that the intended trajectory is being followed.

- Planned obstacle avoidance. Due to the availability of the Jacobian matrix with respect to the trajectory parameters, it becomes straightforward to compute first-order modifications to a planned trajectory which moves it outside the region of intersection with an obstacle.
- Guidepath representation. Many factory automation vehicles express vehicle guidepaths (the robot roads in the factory) in terms of lines and arcs; the more general primitives computed here are a more effective representation.
- Path following. Corrective trajectories for path following applications can be generated in such a manner as to achieve the correct position, heading, and curvature of the point of path re acquisition.

For trajectories which achieve curvature and higher orders of continuity when joined together, prohibitive rapid access storage would be required to implement lookup tables of solutions for a high-density sampling of every terminal pose within a useful range. While interpolation can be used to reduce storage requirements, the algorithm presented here can compute solutions so rapidly from such poor initial estimates that it would often render even tables of initial guesses unnecessary.

While advances in computing continue to render slower algorithms faster, the value of efficient nonholonomic trajectory generation is not restricted to computers of the contemporary generation. In the context of planning around obstacles, where up to thousands of trajectories per second are checked for collisions, an efficient generator enables an efficient planner.

1.4. Prior Work

From a robotics perspective, there has been little work on this problem when compared with, for example, the amount of effort devoted to localization (position estimation). From another, the two-point boundary value problem of differential equations, optimal curves in the calculus of variations, the spline generation problem of geometric modeling, and the optimal control problem are all very closely related and often more general problems.

Some of the earliest work in robotics appeals to the applied mathematics literature to supply a precedent for the problem (Horn 1983; Dubins 1957).

Contemporary applied mathematics literature addresses the relationships between abstract curve generation and control. For example, the relationship between curve fitting and optimal control is addressed in Kano et al. (2003). Here, the use of linear system dynamics to fit curves is first attributed to practitioners in flight control.

Early approaches in robotics are characterized by a curve-fitting formulation where the parameter space of a family of one or more curves of some assumed general form is searched for a solution. With the exception of very early work which used B-splines, researchers initially preferred to represent trajectories in terms of heading, curvature, and higher derivatives, presumably due to their ease of execution and the ease with which curvature constraints can be tested, if not imposed.

The progression is from line segments (Tsumura et al. 1981) to arcs (Komoriya, Tachi, and Tanie 1984) to clothoids (Kanayama and Miyake 1985) to cubic spirals (quadratic curvature) (Kanayama and Hartman 1988). The progression to ever higher-order polynomials reflects the desire to constrain higher-order derivatives for boundary values, and thereby achieve higher levels of continuity when primitives are joined together sequentially.

Aspects of optimization have appeared over time. In Kanayama and Miyake (1985) is an early mention of the nonuniqueness of solutions and searching alternatives. In Kanayama and Hartman (1988) are explicit performance indices and proofs of optimality for clothoids and cubic spirals.

One approach to planning is to sequence atomic primitives together. Dubins (1957) showed that sequences of arcs and lines are shortest for a forward moving vehicle given a constraint on average curvature. Much later, Reeds and Shepp (1990) generalized this result to forward and backward motions, and Shkel and Lumelsky (1996) used classification to improve efficiency.

These works have been restricted to line and arc primitives based on the quest for the shortest path, but in the presence of obstacles or higher-order boundary conditions, more expressive primitives are required. Earlier Shin and Singh (1990) for example, composed trajectories from clothoids and lines for these reasons.

Boisonnat, Cérézo, and Leblond (1992) derived Dubin's result using variational principles. Variational methods are now commonly applied to nonholonomic motion planning in robotics (Latombe 1991; Laumond 1998). The relationship to the two-point boundary value problem has meant that classical numerical methods are applicable. Delingette, Herbert, and Ikeuchi (1991) use a sampled representation of the trajectory and a relaxation based numerical method to compute the optimum input.

The shooting method is another classical technique for solving boundary value and optimal control problems. It is based on assuming a parametric representation and solving for the parameters. The method of substituting a family of curves into a differential equation and solving for the parameters is, of course, classical "variation of parameters". Likewise, the use of power series in order to solve differential equations has been employed for centuries.

From as early as Brockett (1981) it has been known that sinusoidal inputs are optimal from the perspective of minimized squared effort. Murray and Sastry (1993) used this result to

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

generate suboptimal solutions for chained systems. Similarly, Fernandes, Gurvits, and Li (1991) proposed the use of a truncated Fourier series to express the steering input and propose a shooting method to find the coefficients of these sinusoidal basis functions. Polynomial steering functions have been proposed in Tilbury, Murray, and Sastry (1995) for steering in the *n* trailer problem.

Much of the nonholonomic motion planning literature has become theoretical in nature and algorithmic efficiency is not often discussed. In Reuter (1998), however, a near real-time optimal control solution appears. The problem is formulated as eleven simultaneous first-order differential equations subject to boundary conditions which include curvature and its derivative. Solutions are generated in about 1/3 s.

1.5. Approach

The approach presented in this paper is one which combines many of the strengths of earlier techniques in order to achieve both a highly general formulation and a real-time solution. First, the clothoid and related curves of earlier approaches are generalized to a curvature polynomial of arbitrary order which becomes the assumed form of the solution. This form of solution presents several computational advantages. Secondly, the very general optimal control formulation is applied and, based on the assumed form of solution, converted into one of nonlinear programming. Application of numerical methods for nonlinear programming problems then result in solutions for connecting fairly arbitrary postures in under a millisecond of computation.

1.6. Layout

The paper is organized into five sections. In Section 2 we develop the general method for using an assumed solution form to convert the problem from optimal control to constrained optimization. In Section 3 we introduce the polynomial spiral and its properties and develop the computational method of solution. In Section 4 we present the results and in Section 5 the conclusion.

2. Formulation

In this section we formulate trajectory generation first as an optimal control problem, and later as a parametric constrained optimization problem. We then adapt classical numerical methods to the solution.

2.1. Trajectory Generation Problem

The briefest acquaintance with the question of how one determines a steering function which achieves a goal posture leads to the conclusion that the differential equations are unavoidable. The term posture is a convenient generalization of pose which includes curvature and its derivatives. Most generally, control theory provides the notion of state which would include, in this case, velocity and its derivatives.

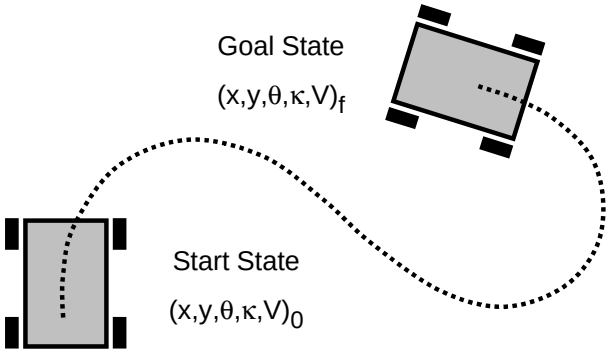

Fig. 3. Trajectory generation problem. The initial and final pose, curvature, velocity and perhaps some of their derivatives are given. The problem is to find an input consistent with all of these constraints, the system dynamics, and bounds on the inputs.

Let the trajectory generation problem be defined here as that of determining a feasible specification of motion which will cause the robot to move from a given initial posture (state) to a given final posture (state). For example, consider the case indicated in Figure 3.

More generally, an ordered list of goal states might be specified and constraints of different forms may apply at each state. While the technique presented in this paper adapts in a straightforward manner to the generation of such dynamic splines, this generalization will not be discussed further.

In the event that more than one feasible motion connects the initial and terminal state, it may be desirable to choose among them based on some convenient performance criterion.

2.2. Optimal Control Formulation

Optimal control is a natural formalism for the representation of such problems. The above problem can be expressed in control theoretic terms as follows.

There are known nonlinear system dynamics:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t). \tag{5}$$

A performance index is expressed as some functional evaluated over the trajectory:

$$J = \phi[x(t_f)] + \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}, t)\mathrm{d}t. \tag{6}$$

There are initial and terminal constraints on the states:

$$\mathbf{x}(t_0) = \mathbf{x}_0 \qquad \mathbf{x}(t_f) = \mathbf{x}_f. \tag{7}$$

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

There may also be certain pragmatic constraints (reflecting such concerns as limited actuator power) on the inputs. For example:

$$|\mathbf{u}(t)| \leq \mathbf{u}_{max}(t) \qquad |\dot{\mathbf{u}}(t)| \leq \dot{\mathbf{u}}_{max}(t). \tag{8}$$

This is a fairly classical formulation of an optimal control problem in the Bolza form.

2.3. Constrained Optimization Formulation

Just as the classical technique of variation of parameters converts differential equations to algebraic ones, it converts optimal control problems to constrained optimization (also known as nonlinear programming) problems. The technique used here is also closely related to the shooting method for boundary value problems because an initial guess will be iteratively refined based on repeated evaluation of the forward solution.

Parametric representations of solutions are convenient from the perspective of the compactness of the representation. When compared with sampled forms of continuous signals, this compactness makes them easier to represent, store, communicate and manipulate. The process of transformation starts by assuming a solution of the form

$$\mathbf{u}(t) = \mathbf{u}(\mathbf{p}, t) \tag{9}$$

where the control is assumed to be a member of a family of functions which span all possible values of an arbitrary vector of parameters $\mathbf{p}$ of length p. An individual control function is now represented as a point in parameter space.

Since the input completely determines the state, and the parameters now determine the input, dependence on both state and input is just dependence on the parameters. Accordingly, the state equations can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(\mathbf{p}, t), \mathbf{u}(\mathbf{p}, t), t] = \mathbf{f}(\mathbf{p}, t). \tag{10}$$

The state vector would include any variables upon which boundary conditions are to be imposed as well as any others upon which these depend.

Let the boundary conditions comprise a set of n constraint relations of the form:

$$\mathbf{h}(\mathbf{p}, t_0, t_f) = \mathbf{x}(t_0) + \int_{t_0}^{t_f} \mathbf{f}(\mathbf{p}, t)\mathrm{d}t = \mathbf{x}_b. \tag{11}$$

It is conventional to write these as:

$$\mathbf{g}(\mathbf{p}, t_0, t_f) = \mathbf{h}(\mathbf{p}, t_0, t_f) - \mathbf{x}_b = 0. \tag{12}$$

Assume there is some scalar performance index which is to be minimized:

$$\text{minimize} : J(\mathbf{x}, \mathbf{u}) = \phi[\mathbf{x}(t_f)] + \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}, t)\mathrm{d}t. \tag{13}$$

Again, because the parameters determine the input which determines the state, these expressions can be written as:

$$\begin{aligned} &\text{minimize} : && J(\mathbf{p}) = \phi(\mathbf{p}, t_f) + \int_{t_0}^{t_f} L(\mathbf{p}, t)\mathrm{d}t \\ &\text{subject to} : && \mathbf{g}(\mathbf{p}, t_0, t_f) = 0 \quad t_f \quad \text{free}. \end{aligned} \tag{14}$$

It may also be useful to represent the constraint of bounded inputs with something like:

$$|\mathbf{p}| \leq \mathbf{p}_{max}. \tag{15}$$

The present problem formulation now looks partially like optimal control (due to the integrals) and partially like parameter optimization. It is now the case that both the state and the performance index are functions only of the parameters and time, but the appearance of integrals in both is nontraditional.

2.4. First-Order Dynamic Response to Parameter Variation

The high-level notation masks some severe difficulties in general. Chief among them is the question of how first derivatives with respect to the parameters are to be determined. There is generally no analytic solution for the state available which can be substituted into the equations in order to compute parameter derivatives of the constraints $\mathbf{g}(\mathbf{q})$. Assuming so begs the original question. Even if the trajectory was available as an integral over the input (as it will be later for polynomial spirals), there is no guarantee that the integrals have closed-form solutions. Hence, implementations generally must rely on numerical methods.

Nonetheless, in order to implement the shooting method, the Jacobians of the performance index and the endpoint with respect to the parameters will be required. The Leibnitz rule supplies the principle but the Jacobians can only be extracted indirectly.

The system dynamics can be differentiated with respect to the parameters to obtain

$$\frac{\partial}{\partial \mathbf{p}}\dot{\mathbf{x}}(t) = \left[\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{p}}\right] = F(\mathbf{p}, t)\frac{\partial \mathbf{x}}{\partial \mathbf{p}} + G(\mathbf{p}, t)\frac{\partial \mathbf{u}}{\partial \mathbf{p}}$$

$$F = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \qquad G = \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \tag{16}$$

where F and G are the Jacobians of the system with respect to the state and the inputs. Hence, the Jacobian of the state with respect to the parameters must satisfy the linearized system dynamics at any point in time. Although there is no solution to the nonlinear system given in eq. (1), the general solution to the linearized dynamics (which must exist due to linearity) exists in closed form (Kelly 2001).

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

Integrating the above gives a self-referential form of the Leibnitz' rule:

$$\frac{\partial \mathbf{x}}{\partial \mathbf{p}} = \int_{t_0}^{t_f} \left[F(\mathbf{p}, t) \frac{\partial \mathbf{x}}{\partial \mathbf{p}} + G(\mathbf{p}, t) \frac{\partial \mathbf{u}}{\partial \mathbf{p}} \right] \mathrm{d}t. \tag{17}$$

This equation can be integrated to yield the parameter Jacobian at the endpoint. Due to the nonlinearity of the equations, a solution would proceed by linearizing the first-order conditions. Therefore, second derivatives would be required in general.

In some situations, it is possible to eliminate the self reference to the state to produce a solution integral (a quadrature) which is explicitly of the form:

$$\mathbf{x}(\mathbf{p}, t_f) = \int_{t_0}^{t_f} \mathbf{g}(\mathbf{p}, t) \mathrm{d}t. \tag{18}$$

The Leibnitz rule can be applied directly to this form to get the parameter Jacobian and the final time gradient:

$$\frac{\partial \mathbf{x}}{\partial \mathbf{p}} = \int_{t_0}^{t_f} \left[\frac{\partial \mathbf{g}}{\partial \mathbf{p}} \right] \mathrm{d}t \qquad \frac{\partial \mathbf{x}}{\partial t_f} = \mathbf{g}(\mathbf{p}, t_f). \tag{19}$$

Even in this simplified case, however, the parameter Jacobian remains defined by an integral.

Likewise, the performance index can be differentiated:

$$\frac{\partial}{\partial \mathbf{p}} J(\mathbf{p}) = \frac{\partial}{\partial \mathbf{p}} \phi(\mathbf{p}, t_f) + \int_{t_0}^{t_f} \left\{ \frac{\partial}{\partial \mathbf{x}} L(\mathbf{p}, t) \frac{\partial \mathbf{x}}{\partial \mathbf{p}} + \frac{\partial}{\partial \mathbf{u}} L(\mathbf{p}, t) \frac{\partial \mathbf{u}}{\partial \mathbf{p}} \right\} \mathrm{d}t. \tag{20}$$

This information is included here to address how the approach applies to any parametric form of assumed solution. The choice of polynomial spiral will further simplify the calculations, but the overall approach can be applied to any form of parametric input.

2.5. Functional Inequality Constraints

Once the constrained optimization formulation is adopted, systematic mechanisms for dealing with inequality constraints, such as in Kuhn and Tucker (1961) can be brought to bear. Imposing limits on an input during iteration can be problematic because such limits apply to the entire time history of the input. Preventing excessive curvature at one time for example, may cause it somewhere else.

An auxiliary advantage of the present formulation is that the maximum value of an input can be approximated by computing its n-norm

$$\max\{u_i(t)^n\} \approx \frac{1}{(t_f - t_0)} \int_{t_0}^{t_f} [u_i(t)]^n \mathrm{d}t \tag{21}$$

where n is a large even integer. This expression can now form the basis of an integral constraint

$$\max\{u_i(t)^n\} \leq (u_{max})^n \tag{22}$$

which is no different in principle than the boundary conditions except that it is an inequality.

2.6. Change of Variable

It is often convenient for trajectory generation purposes to change the independent variable from time to distance. Without loss of generality, let the initial distance be set to zero. Also, let the final distance s_f be absorbed into an adjoined parameter vector thus:

$$\mathbf{q} = \left[\mathbf{p}^{\mathrm{T}}, s_f \right]^{\mathrm{T}}. \tag{23}$$

The problem formulation under this change of variable takes the following form:

$$\begin{aligned} &\text{minimize :} && J(\mathbf{q}) = \phi(\mathbf{q}) + \int_0^{s_f} L(\mathbf{q}) \mathrm{d}s \\ &\text{subject to :} && \mathbf{g}(\mathbf{q}) = 0 \quad s_f \quad \text{free.} \end{aligned} \tag{24}$$

2.7. Necessary Conditions

Constrained optimization problems can of course be solved by the method of Lagrange multipliers. From the theory of constrained optimization, the solution is obtained by defining the Hamiltonian (often called the Lagrangian in the constrained optimization context):

$$H(\mathbf{q}, \boldsymbol{\lambda}) = J(\mathbf{q}) + \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{g}(\mathbf{q}). \tag{25}$$

The first-order necessary conditions are:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{q}} H(\mathbf{q}, \boldsymbol{\lambda}) &= \frac{\partial}{\partial \mathbf{q}} J(\mathbf{q}) + \boldsymbol{\lambda}^{\mathrm{T}} \frac{\partial}{\partial \mathbf{q}} \mathbf{g}(\mathbf{q}) = \mathbf{0}^{\mathrm{T}} \quad (p + 1 \text{ eqns}) \\ \frac{\partial}{\partial \boldsymbol{\lambda}} H(\mathbf{q}) &= \mathbf{g}(\mathbf{q}) = \mathbf{0} \qquad (n \text{ eqns}). \end{aligned} \tag{26}$$

This is a set of $n + p + 1$ simultaneous equations in the $n + p + 1$ unknowns in $\mathbf{q}$ and $\boldsymbol{\lambda}$. The equation in the first set corresponding to the final distance is the transversality condition used to determine the final distance in the event that it is considered variable. The right-hand side is a row vector in the first set and a column vector in the second set.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

2.8. Methods of Solution

Several numerical techniques are available for the solution of nonlinear programming problems. The last five chapters of Luenberger (1989), for example, contain a lucid tutorial on the basic techniques: gradient projection, Lagrange, and penalty function.

The technique chosen here is a Lagrange method. Such methods treat the Lagrange multipliers on an equal footing with the unknown parameters. The first-order necessary conditions are solved directly using multi-dimensional rootfinding techniques. Newton's method is the basis for most of these "curvature" (second derivative) based techniques. As a result, they converge quadratically but must be augmented by mechanisms to enhance stability when operating far from a solution.

Newton's method as it applies to constrained optimization is derived briefly as follows. Transposing the first set of equations, linearizing about a point where all equations are not satisfied, and insisting that they become satisfied to first order after perturbation gives:

$$\frac{\partial^2 H}{\partial \mathbf{q}^2}(\mathbf{q},\boldsymbol{\lambda})\Delta\mathbf{q} + \frac{\partial}{\partial \mathbf{q}}\mathbf{g}(\mathbf{q})^{\mathrm{T}}\Delta\boldsymbol{\lambda} = -\frac{\partial}{\partial \mathbf{q}}H(\mathbf{q},\boldsymbol{\lambda})^{\mathrm{T}} \quad (p+1 \text{ eqns})$$

$$\frac{\partial}{\partial \mathbf{q}}\mathbf{g}(\mathbf{q})\Delta\mathbf{q} = -\mathbf{g}(\mathbf{q}) \qquad (n \text{ eqns}). \tag{27}$$

Notation for the Hessian of the Hamiltonian (with respect to $\mathbf{q}$) was used:

$$\frac{\partial^2 H}{\partial \mathbf{q}^2}(\mathbf{q},\boldsymbol{\lambda}) = \frac{\partial^2 J}{\partial \mathbf{q}^2}(\mathbf{q}) + \boldsymbol{\lambda}\frac{\partial^2}{\partial \mathbf{q}^2}\mathbf{g}(\mathbf{q}). \tag{28}$$

The last term involves a third-order tensor. It can be interpreted as a multiplier weighted sum of the Hessians of each of the individual constraint equations:

$$\boldsymbol{\lambda}\frac{\partial^2}{\partial \mathbf{q}^2}\mathbf{g}(\mathbf{q}) = \sum_i \lambda_i \frac{\partial^2}{\partial \mathbf{q}^2} g_i(\mathbf{q}). \tag{29}$$

In matrix form, this is now of the form:

$$\begin{bmatrix} \dfrac{\partial^2 H}{\partial \mathbf{q}^2}(\mathbf{q},\boldsymbol{\lambda}) & \dfrac{\partial}{\partial \mathbf{q}}\mathbf{g}(\mathbf{q})^{\mathrm{T}} \\ \dfrac{\partial}{\partial \mathbf{q}}\mathbf{g}(\mathbf{q}) & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{q} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\dfrac{\partial}{\partial \mathbf{q}}H(\mathbf{q},\boldsymbol{\lambda})^{\mathrm{T}} \\ -\mathbf{g}(\mathbf{q}) \end{bmatrix}. \tag{30}$$

This matrix equation can be interpreted to provide the errors in the parameters which, when added to the parameters, remove the observed residuals to first order. It can be iterated from a good initial guess for the parameters and the multipliers until convergence.

It is usually advisable to enforce diagonal dominance in the manner of the Levenberg–Marquardt algorithm (see Press 1988) in order to enlarge the radius of convergence. The structure is also amenable to recursive partitioning (Slama, Theurer, and Henriksen 1980) to improve performance.

Two degenerate forms of the iteration are also important for this paper.

2.8.1. Unconstrained Optimization

The degenerate form of unconstrained optimization is of practical significance. In this case, there are no constraint equations and no Lagrange multipliers. Equation (30) degenerates to the square system

$$\left[\frac{\partial^2 J}{\partial \mathbf{q}^2}(\mathbf{q})\right]\Delta\mathbf{q} = -\frac{\partial}{\partial \mathbf{q}}J(\mathbf{q})^{\mathrm{T}} \tag{31}$$

which is the same result that would be obtained by applying Newton's method from scratch to this specific problem.

2.8.2. Constraint Satisfaction

The second degenerate form is also of practical significance. Indeed, this is trajectory generation as it was originally posed. Here, there is no objective function. It is simply required that the boundary conditions be met. In this case, eq. (30) degenerates to:

$$\left[\frac{\partial}{\partial \mathbf{q}}\mathbf{g}(\mathbf{q})\right]\Delta\mathbf{q} = -\mathbf{g}(\mathbf{q}). \tag{32}$$

This is Newton's method as it occurs in rootfinding contexts. The matrix which appears is the Jacobian of the constraints. In this case, it is legitimate for the system to be non-square and, if it is, the appropriate generalized inverse of the Jacobian can be used in the iteration.

3. Solution Using Polynomial Spirals

The techniques of the previous section will apply to any form of assumed solution. In this section we develop the specific case when polynomial spirals are the assumed form.

Furthermore, the previous formulation is not confined to steering functions. It could be used, for example, to determine polynomials for linear velocity and to enforce acceleration continuity. However, such a trivial problem would not require all of the machinery just presented. Since steering functions are much harder to determine, in this section we concentrate on this aspect of the problem.

3.1. Achieving Steering Continuity

Even though it is most accurate to consider curvature a state, it is also useful for many purposes to consider curvature to be an input and to omit the last equation in eq. (1). Under this assumption, the equations become homogeneous to the first degree in linear velocity $V(t)$ and it becomes possible to divide the remaining equations by it in the form of $\mathrm{d}s/\mathrm{d}t$ to

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

effect a change of independent variable from time to distance:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}s}x(s) &= \cos\theta(s) \\ \frac{\mathrm{d}}{\mathrm{d}s}y(s) &= \sin\theta(s) \\ \frac{\mathrm{d}}{\mathrm{d}s}\theta(s) &= \kappa(s). \end{aligned} \tag{33}$$

This form can be most useful for trajectory generation purposes because it permits the geometry of the trajectory to be considered independent of the speed of traversal.

Under this transformation, and omitting velocity, the endpoint constraints in Figure 3 become:

$$\begin{aligned} \mathbf{x}(s_0) &= (x_0, y_0, \theta_0, \kappa_0)^{\mathrm{T}} \\ \mathbf{x}(s_f) &= (x_f, y_f, \theta_f, \kappa_f)^{\mathrm{T}}. \end{aligned} \tag{34}$$

Of course, if the origin is defined to be positioned at the initial posture then the first three boundary conditions will be satisfied by construction and the five constraints $[\kappa_0, x_f, y_f, \theta_f, \kappa_f]$ would remain to be satisfied in this example. Five parameters are therefore necessary for generating sequences of curvature continuous trajectories.

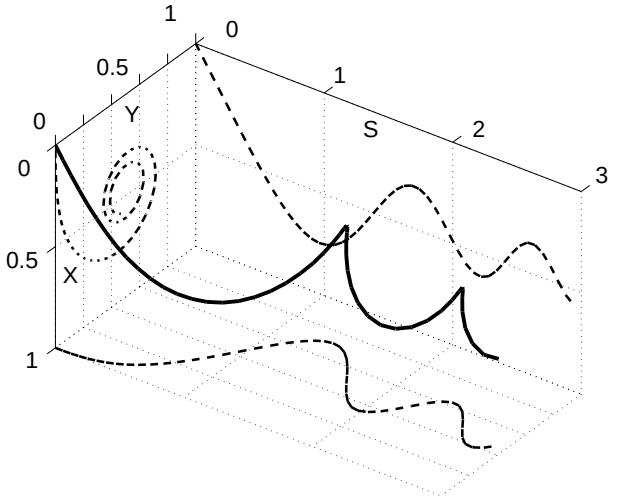

Fig. 4. Cornu spiral. The projection onto the complex plane is the plane trajectory which results from a linearly varying curvature input. The curvature polynomial coefficients are $b = 0$, and $c = \pi$.

3.2. Clothoids

The clothoid is a well-known curve which is defined by linearly varying curvature, thus

$$\kappa(s) = a + bs. \tag{35}$$

This curve traces out a trajectory in x–y–s space known as the Cornu spiral (see Figure 4). Of course, with only three parameters to vary (a, b and s) the clothoid cannot satisfy arbitrary terminal curvature or even heading constraints if the existing parameters are used to satisfy initial curvature and terminal position.

3.3. Polynomial Spirals

Given the need for additional parameters for steering continuity, an obvious approach is to add terms to the curvature polynomial. These curves, called polynomial spirals in Dillen (1990), possess as many degrees of freedom as necessary to meet any number of constraints. An nth-order spiral is simply an nth-order polynomial expressing curvature in terms of arc length:

$$\kappa(s) = a + bs + cs^2 + ds^3 + \ldots . \tag{36}$$

In the complex plane, these curves will be referred to as the *generalized Cornu spiral*. A representative spiral of cubic order is shown in Figure 5.

This new primitive possesses many advantages that can be briefly summarized as the ability to represent any feasible

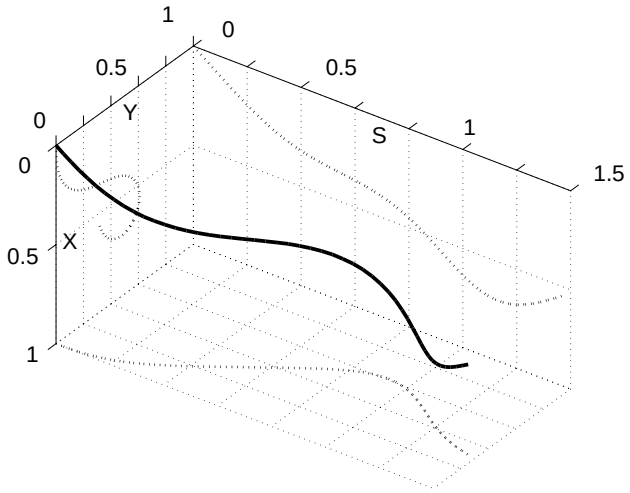

Fig. 5. Generalized Cornu spiral. This curve is generated by a cubic curvature polynomial of the form $\kappa(s) = a + bs + cs^2 + ds^3$. The coefficients are $a = 0.0$, $b = 33$, $c = -82$, and $d = 41.5$. In this example, the primitive reverses curvature and terminates pointing back at the origin.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

vehicle motion using a small number of parameters. Such a bold statement is easy to justify by noting that all feasible motions have an associated control and the primitive is merely the Taylor series of the control. The Taylor remainder theorem then supplies the basis of the claim that all controls can be represented.

Given that the control in this case represents the actual motion of the steering actuator, it can also be concluded that higher-order terms in the series will eventually vanish due to the impossibly high frequencies that they imply. A small number of parameters is also valuable from the perspective of representing and communicating the results, but most importantly, it dramatically reduces the dimensionality of the search space which is implied in all variational approaches to the problem.

3.4. Reduction to Decoupled Quadratures

The polynomial spiral has two other important computational advantages. Notice that the system dynamics, while coupled, are in echelon form, so that a closed-form solution for heading could be substituted into the position integrals to decouple the system.

The polynomial spiral can be integrated in closed form to produce heading:

$$\begin{aligned} \kappa(s) &= a + bs + cs^2 + ds^3 + \dots \\ \theta(s) &= as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \dots . \end{aligned} \tag{37}$$

The new position integrals then become:

$$\begin{aligned} x(s) &= \int_0^s \cos\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \dots\right] \mathrm{d}s \\ y(s) &= \int_0^s \sin\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \dots\right] \mathrm{d}s. \end{aligned} \tag{38}$$

These are the generalized Fresnel integrals. The computation of these transcendental integrals and their gradients with respect to the parameters will turn out to be the major computational burden of trajectory generation.

The main advantage of decoupling is that the first-order behavior of the system can now also be computed using quadrature rather than something like the multidimensional Runge–Kutta required by eq. (17). The parameter Jacobians remain defined by integrals but at least they are explicit. Their integral nature also leads to an interpretation of eq. (32) as the classical shooting method for solving boundary value problems.

A second computational advantage is that the state equations become simplified to the maximum degree possible while retaining a fairly general steering function. It is well known that the position equations are not solvable in closed form for even a linearly varying heading input (in which case the integrals are the well-known Fresnel integrals). However, the polynomial spiral has the property that any number of boundary conditions on initial or terminal heading or its derivatives are linear in all parameters but distance. This means that it is straightforward to enforce these conditions exactly by fixing one parameter in addition to terminal distance and solving the resulting linear equations; only two of the equations are ever difficult to solve.

3.5. Boundary Conditions as Constraints

Consider now the expression of the boundary conditions for initial and terminal posture using polynomial spirals for the case of enforcing curvature continuity. This case corresponds to cubic polynomials and five parameters:

$$\kappa(s) = a + bs + cs^2 + ds^3. \tag{39}$$

The initial constraints on position and heading can be satisfied trivially by choosing coordinates such that:

$$s_0 = 0 \qquad x(0) = y(0) = \theta(0) = 0. \tag{40}$$

This leaves constraints on initial curvature and its derivatives as well as the entire final posture to be satisfied. We define the vector of polynomial spiral parameters to be the coefficients:

$$\mathbf{q} = \begin{bmatrix} a & b & c & d & s_f \end{bmatrix}^{\mathrm{T}}. \tag{41}$$

The initial curvature is satisfied trivially:

$$a = \kappa(0) \tag{42}$$

and any higher-order initial derivatives in a more general case could be resolved similarly. To save computation, this parameter will be eliminated from the iterative equations. The remaining constraint equations in standard form are therefore:

$$\mathbf{g}(\mathbf{q}) = \mathbf{h}(\mathbf{q}) - \mathbf{x}_b = 0 \quad \text{or} \quad \begin{matrix} h_1(\mathbf{q}) - x_f = 0 \\ h_2(\mathbf{q}) - y_f = 0 \\ h_3(\mathbf{q}) - \theta_f = 0 \\ h_4(\mathbf{q}) - \kappa_f = 0 \end{matrix}. \tag{43}$$

The equations for terminal curvature κ_f and heading θ_f are polynomials while the endpoint constraints x_f and y_f are quadratures. The Jacobian matrix of the above nonlinear system is just a top to bottom arrangement of the gradients of each constraint:

$$\frac{\partial \mathbf{g}}{\partial \mathbf{q}} = \frac{\partial \mathbf{h}}{\partial \mathbf{q}} = \begin{bmatrix} \dfrac{\partial x}{\partial b} & \dfrac{\partial x}{\partial c} & \dfrac{\partial x}{\partial d} & \cdots & \dfrac{\partial x}{\partial s_f} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial \kappa}{\partial b} & \dfrac{\partial \kappa}{\partial c} & \dfrac{\partial \kappa}{\partial d} & \cdots & \dfrac{\partial \kappa}{\partial s_f} \end{bmatrix}. \tag{44}$$

For the position coordinates, the Leibnitz' rule can be used to compute the derivatives. Details of a Simpson's rule implementation for computing these functions and their first and second derivatives are provided in the Appendix.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

3.6. Boundary Conditions as Performance Indices

It is also valid and may be convenient to formulate the boundary conditions as costs rather than hard constraints. Let the terminal error be scaled as follows:

$$\Delta\mathbf{x}(\mathbf{q}) = \begin{bmatrix} x(\mathbf{q}) - x_f \\ y(\mathbf{q}) - y_f \\ L[\theta(\mathbf{q}) - \theta_f] \\ L^2[\kappa(\mathbf{q}) - \kappa_f] \end{bmatrix}. \tag{45}$$

The characteristic length L is used to make the units of each element of the vector consistent and to make each element of roughly equal significance. A useful interpretation is that the heading and curvature error are being converted to their effect on the position of a point a distance L away. For the forktruck application, for example, L could be set to the sum of the length of the forks and the distance the vehicle will travel to insert them. The absolute bounds on this terminal error are denoted as:

$$\Delta\mathbf{x}_{max} = [\Delta x_{max}\, \Delta y_{max}\, \Delta\theta_{max}\, \Delta\kappa_{max}]^{\mathrm{T}}. \tag{46}$$

An associated performance index could then given by the squared terminal error:

$$\phi(\mathbf{q}) = \frac{1}{2}\left[\Delta\mathbf{x}(\mathbf{q})\right]^{\mathrm{T}}[\Delta\mathbf{x}(\mathbf{q})]. \tag{47}$$

The gradient of the performance index with respect to the parameters is the sum of several components:

$$\frac{\partial\phi}{\partial\mathbf{q}} = \left[\frac{\partial x}{\partial\mathbf{q}}\Delta x + \frac{\partial y}{\partial\mathbf{q}}\Delta y + L^2\frac{\partial\theta}{\partial\mathbf{q}}\Delta\theta + L^4\frac{\partial\kappa}{\partial\mathbf{q}}\Delta\kappa\right]. \tag{48}$$

Here, the vectors $\partial x/\partial\mathbf{q}$, etc., are the gradients (row vectors) of the terminal posture with respect to the parameters. The Hessian matrix is computed by differentiating this:

$$\frac{\partial^2}{\partial\mathbf{q}^2}(\phi) = \left[\frac{\partial^2 x}{\partial\mathbf{q}^2}\Delta x + \frac{\partial^2 y}{\partial\mathbf{q}^2}\Delta y + L^2\frac{\partial^2\theta}{\partial\mathbf{q}^2}\Delta\theta + L^4\frac{\partial^2\kappa}{\partial\mathbf{q}^2}\Delta\kappa + \frac{\partial x^{\mathrm{T}}}{\partial\mathbf{q}}\frac{\partial x}{\partial\mathbf{q}} + \frac{\partial y^{\mathrm{T}}}{\partial\mathbf{q}}\frac{\partial y}{\partial\mathbf{q}} + L^2\frac{\partial\theta^{\mathrm{T}}}{\partial\mathbf{q}}\frac{\partial\theta}{\partial\mathbf{q}} + L^4\frac{\partial\kappa^{\mathrm{T}}}{\partial\mathbf{q}}\frac{\partial\kappa}{\partial\mathbf{q}}\right]. \tag{49}$$

The last four terms involve outer product (hence symmetric) matrices formed by multiplying individual gradients by their transposes. These outer products can be evaluated numerically given the gradients. When near a solution, these outer product terms dominate the Hessian.

3.7. Smoothness as a Performance Index

It is also possible to create a performance index which prefers smooth trajectories. In this case, an integral form is appropriate. The following functional form will discourage high curvature values relative to more graceful turns:

$$J_\kappa(\mathbf{q}) = \frac{1}{2}\int_0^{s_f}[\kappa(\mathbf{q})]^2 \mathrm{d}s. \tag{50}$$

Terms involving heading or curvature derivatives could also be added to discourage indirect routes or rapid steering changes. Note that whenever $\mathbf{q}$ appears inside a distance integral, the terminal arc length s_f should be interpreted as the variable of integration s.

The gradient of the performance index with respect to the parameters is obtained from the Leibnitz' rule:

$$\frac{\partial J_\kappa}{\partial\mathbf{q}} = \int_0^{s_f}\kappa(\mathbf{q})\frac{\partial\kappa}{\partial\mathbf{q}}\mathrm{d}s. \tag{51}$$

The Hessian matrix is computed by differentiating this:

$$\frac{\partial^2}{\partial\mathbf{q}^2}(J_\kappa) = \int_0^{s_f}\left[\frac{\partial\kappa^{\mathrm{T}}}{\partial\mathbf{q}}\frac{\partial\kappa}{\partial\mathbf{q}} + \kappa(\mathbf{q})\frac{\partial^2\kappa}{\partial\mathbf{q}^2}\right]\mathrm{d}s. \tag{52}$$

The previous performance index could also be added to this one in order to produce smooth trajectories which almost meet the constraints in some optimum fashion.

4. Results

In this section we present some numerical validations of the approach to trajectory generation.

4.1. Forward Problem

A good solution to the forward problem for the boundary conditions and their derivatives is necessary because it becomes the basis for solving the more difficult inverse problem. Of course, only the position coordinates present any difficulty.

In the present implementation, Simpson's rule is used to perform all of the integrations numerically. Many of the integrands are quite smooth and can often be estimated well numerically in as little as ten integrand evaluations. In this context, "estimated well" means well enough to determine end posture to perhaps a few millimeters in position and a few milliradians in heading.

When coefficients are large in magnitude, it may be necessary to perform many integrand evaluations in Simpson's rule. At some point, large coefficients correspond to infeasible inputs and the difficulty in computation indicates difficulty or even impossibility of execution. In rough terms, curves that cannot be computed, cannot be executed anyway. The appendix provides a straightforward mechanism to reuse computations while refining the estimate of the quadratures.

Figure 6 shows a typical member of the cubic curvature polynomial family of curves computed. Curves must be much

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

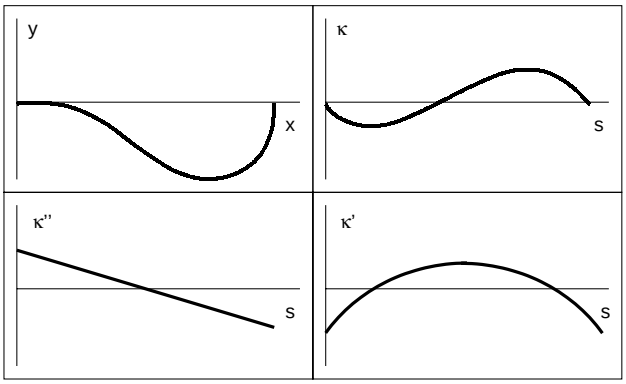

Fig. 6. A cubic polynomial spiral. Clockwise from top left, a curve of cubic polynomial curvature, its curvature, and the first and second derivatives of curvature versus distance.

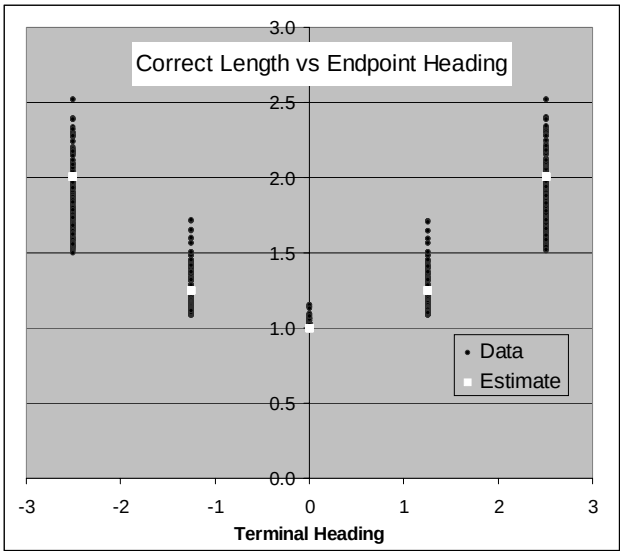

Fig. 7. Approximate length. Over the range of terminal postures discussed in Section 4.3, the correct curve length can be approximated by a quadratic in terminal heading.

more aggressive than this one to present difficulty in the forward solution.

4.2. Initialization

All forms of the inverse solution require an initial estimate in order to start the iterations. Estimates must be very good to have a significant impact on the efficiency because most of the time is spent fine tuning the answer. Therefore, efficiency is not a strong motivation to produce good initial estimates.

However, initialization is also an important matter because the algorithm is convergent only to the nearest local extremum. One extreme on the spectrum of initialization solutions is to use low resolution lookup tables to seed the search for a solution. The other extreme is to use approximations to the solution. The second option has turned out easiest to do well enough for the present purpose.

There is potential value in using scaling and symmetry to reduce the number of solutions being represented. For a given polynomial spiral, a new curve which is scaled by a factor of λ from the original has coefficients:

$$a_{new} = \frac{a}{\lambda} \quad b_{new} = \frac{b}{\lambda^2} \quad c_{new} = \frac{c}{\lambda^3} \quad d_{new} = \frac{d}{\lambda^4}. \tag{53}$$

Reflections about the x- or y-axis or the origin can be accomplished in a similar manner.

If the scale factor applied is the inverse of the original length or endpoint radius, the result is a canonical "unity scale" curve. For the constraint satisfaction case discussed next, our approach is to scale the problem so that the endpoint is on the unit circle. Then, the simple estimate for length illustrated in Figure 7 is used.

$$S \approx \frac{\theta^2}{5} + 1 \tag{54}$$

Any other parameter can then be set to zero and the remaining parameters can be determined by solving the non-position constraints exactly. This procedure enables a robust convergence to a solution over the wide range of cases discussed next because it confines the iterations to a well-behaved subspace of parameter space. In a movie of the iteration taking place, the terminal heading and curvature are always correct and the iteration progressively moves the endpoint to the solution position.

For the constrained and unconstrained optimization cases discussed later, the fully determined case is solved first in order to serve as an initial estimate. Initial estimates for the Lagrange multipliers can be obtained by solving the first part of equation (26) with the pseudoinverter.

4.3. Constraint Satisfaction

The implementation of basic trajectory generation has been highly successful. This case is characterized by the use of eq. (32). In order to assess efficiency and reliability, 1600 solutions were computed over the envelope:

$$5m < x_f < 15m \qquad -5m < y_f < 5.0m$$

$$-\frac{4\pi}{5} < \theta_f < \frac{4\pi}{5}$$

$$-0.1\frac{1}{m} < \kappa_0 < 0.1\frac{1}{m} \qquad -0.1\frac{1}{m} < \kappa_f < 0.1\frac{1}{m}. \tag{55}$$

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

This posture envelope is intended to be representative of a factory automation application (no high curvatures). Termination was based on achieving a weighted residual norm of 0.01 defined as:

$$r = \sqrt{(w_x \Delta x_f)^2 + (w_y \Delta y_f)^2 + (w_\theta \Delta \theta_f)^2 + (w_\kappa \Delta \kappa_f)^2}. \quad (56)$$

The weights were adjusted so that 0.01 units of position or 0.0001 rads of heading or 0.0001 rads/meter of curvature error alone would exceed the threshold. Figure 8 depicts some example curves in the envelope computed.

Computation times over this envelope are summarized in Figure 9. These are essentially worst-case results because no lookup tables and only the above simplistic initial estimate (an arc of roughly the right length) were used. Of course, many applications would sequentially compute many nearby trajectories and run-times would be significantly faster.

Solutions for driving in reverse are generated naturally by choice of sign of the initial distance estimate. Reversing both x and distance in the above cases generates the y-axis mirror image of the solutions. It is also relatively easy to control the number of loops and the direction to which the initial turn will tend. Figure 10, for example, shows two solutions separated by one revolution that were generated by simply modulating the terminal heading.

4.4. Unconstrained Optimization

This case is characterized by the use of eq. (31). This is a fairly general technique in its own right because it is equivalent to the penalty function approach to constrained optimization; we simply convert the constraints to costs with high associated weights. There are always the right number of equations no matter how many curve parameters are used. Figure 11 illustrates an example motivated by the forktruck application. A load is discovered to be 5 m to the right of expectations when only 5 m away. A new trajectory must be generated to move diagonally but end up at zero heading and curvature.

When there is one parameter too few, the weights are configured to ignore terminal curvature and the goal pose is achieved with nonzero terminal curvature (see Figure 12). The five-parameter case computes the same answer as constraint satisfaction.

In the eight-parameter case, the path smoothness performance index is introduced and weights are adjusted to permit terminal heading and curvature error. In this case, the smoothness (total area under squared curvature) is enhanced significantly because the temporarily high initial curvature is balanced by a long intermediate period of low curvature.

4.5. Constrained Optimization

This case is defined by the use of eq. (30). Figures 13 and 14 present results on a curvaceous trajectory chosen so as to

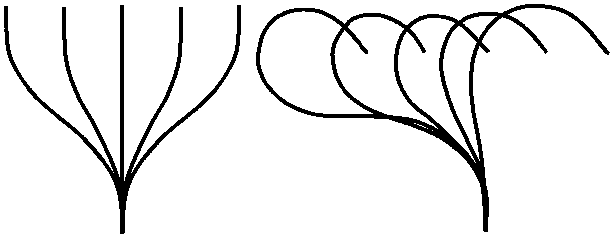

Fig. 8. Example polynomial spirals. These different end postures have feasible solutions.

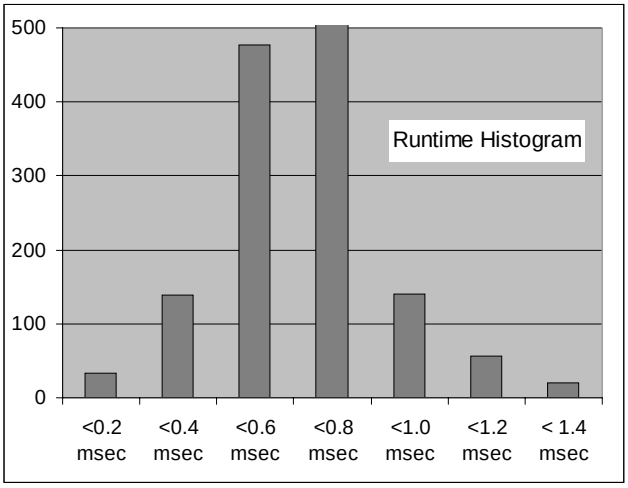

Fig. 9. Cubic polynomial spiral run times. Run times on a 1 GHz Pentium 4 for 1600 constraint satisfaction test cases. All were less than 1.4 ms based on an arc initial estimate.

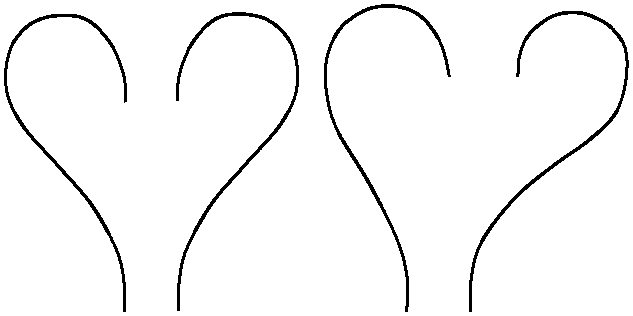

Fig. 10. Multiple solutions. Two different symmetric curves (left) and asymmetric curves (right) each reaching the same relative destination posture.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

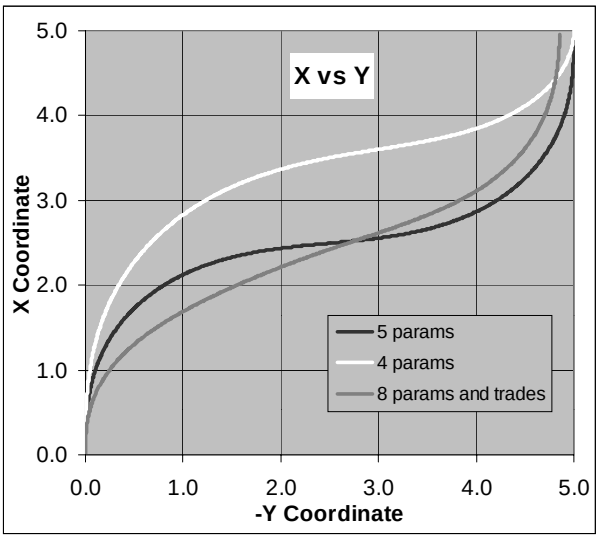

Fig. 11. Trajectories computed using unconstrained optimization. These curves illustrate the use of eq. (31). The goal is to move forward and to the right and end up facing forward with zero curvature. The white curve has too few parameters; the black is exact; the gray trades endpoint error for smoothness.

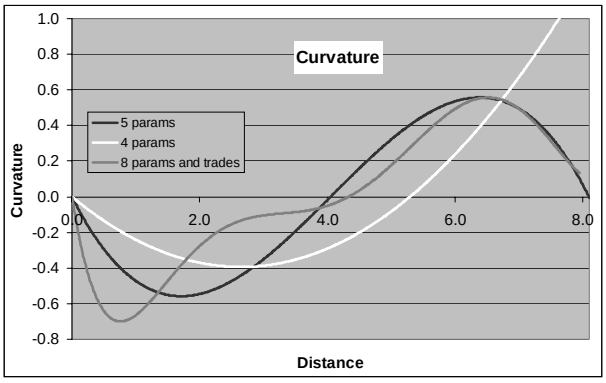

Fig. 12. Curvature profiles for trajectories computed using unconstrained optimization. The four-parameter case does not achieve the desired terminal curvature of zero. The eight-parameter case has the best smoothness index.

illustrate the operation. The trajectory has no initial curvature and is required to terminate at the posture:

$$[x, y, \theta, \kappa] = \left[5, 0, \frac{3\pi}{4}, 0\right]. \tag{57}$$

At least five parameters are required to satisfy the boundary conditions. The five-parameter solution therefore retains no free parameters that can be used to improve the performance

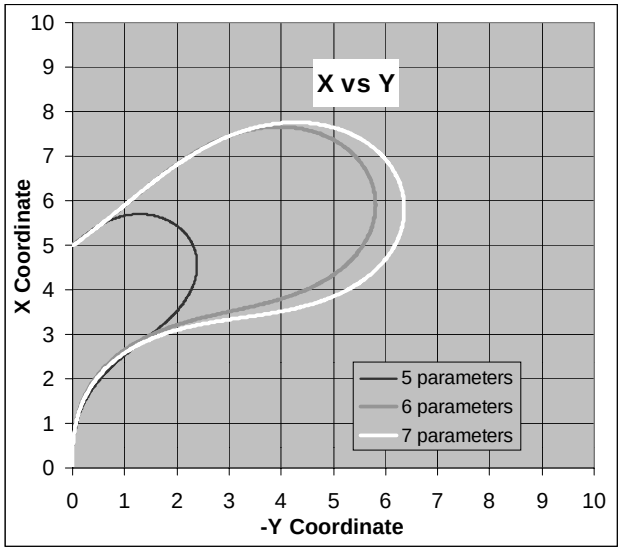

Fig. 13. Trajectories for varying number of parameters. These three curves all terminate at the same posture but smoothness increases with the number of parameters used.

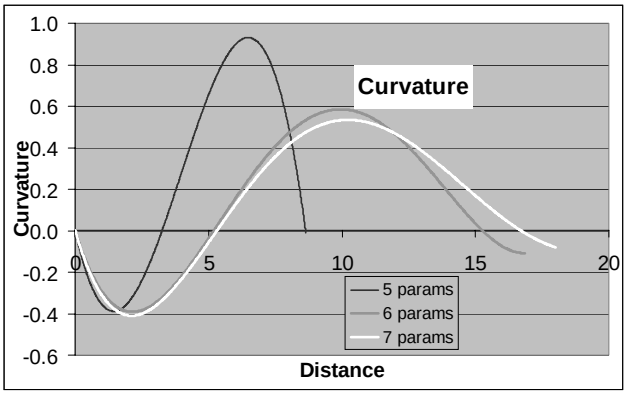

Fig. 14. Curvature profiles for varying number of parameters. These three curves all terminate at the same posture but smoothness increases with the number of parameters used.

index. When free parameters become available (in the six- and seven-parameter solutions), there is an initial dramatic effect on the curvatures used but the effect tapers off quickly.

The performance index used is essentially the continuous inner product of curvature with itself. This index will favor longer paths over shorter ones to the degree that the area under the squared curvature function is reduced by doing so. Of course, lengthening itself has an impact on the performance index so a best compromise is sought. The curvature profiles in Figure 14 show how longer paths can achieve lower overall curvature.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

In practice, different performance indices may be more appropriate. Given the need for a numerical implementation, however, any index at all could be accommodated instead of the example used.

5. Summary and Conclusions

Work in contemporary applications points to an acute need for a mobile robot to understand its own motion capabilities and limitations in very certain terms. This is especially so in the case when a changing or unknown environment requires on-line selection or computation of trajectories.

If trajectories are to be pre-stored, the number of continuous feasible curvature profiles is prohibitively large to represent. Parametric trajectory representations must be used. Even if trajectories are computed on-line rather than off-line, parametric representations then become motivated by the need to compute answers in real time. When they are used, the best parametrizations can represent essentially all feasible motions in the least number of parameters.

Once parametric representations are accepted, the nonlinear nature of the dynamics then requires search in order to find a solution to the inverse problem. A method to compute the first-order response of the dynamics to parameter variation is then required. In general, the parametric gradient of the dynamics can be obtained by linearizing the dynamics with respect to the parameters and solving the associated forward problem (by solving a matrix differential equation).

It is therefore generally possible to convert an optimal control problem into a derived nonlinear programming problem which finds the parameters of the solution curve. Numerical solution of the associated first-order necessary conditions requires the forward solution of auxiliary differential equations, but otherwise, classical numerical approaches apply directly.

Within this general framework, the polynomial spiral is introduced as a primitive which spans the set of feasible controls while using few parameters. It also has a closed form solution for heading which decouples the dynamics and reduces the auxiliary differential equations to decoupled quadratures. This primitive is not only natural to manipulate but it is easy to compute relative to the alternatives. It even reduces lookup table memory requirements because all but two parameters are solvable in closed form.

Based on the use of this primitive to represent arbitrary steering functions, a robust technique has been presented which computes curvature continuous trajectories between relatively arbitrary initial and final postures in under a millisecond on contemporary processors.

Given nonlinear performance indices and/or nonlinear constraints, the algorithm will converge only to the nearest local optimum. This is an inevitable limitation of all local methods. Techniques for generating a good initial guess may therefore be important in cases more ambitious than our examples.

Of course, solutions are only "nearly" optimum in the sense of Fernandes, Gurvits, and Li (1991). A truncated series approximation may exclude a better solution which happens to have non-vanishing high-order derivatives.

In this paper, we have avoided the discussion of how an ordered sequence of goal postures can be achieved using nonlinear programming but extensions in this direction are straightforward.

Appendices

For readers who may be interested in implementing the technique of the article, this appendix provides further detail on the numerical implementation.

Appendix A: Partial Derivatives of Solution Quadratures

Consider the form of the first and second partial derivatives of:

- the solution integrals for position;
- the heading and curvature polynomials;
- a smoothness performance index.

Recall the integral of the state equations expressed as decoupled quadratures:

$$\begin{aligned} x(\mathbf{p}, s_f) &= \int_0^s \cos\theta(\mathbf{p}, s)\mathrm{d}s \quad \theta(\mathbf{p}, s_f) = \int_0^s \kappa(\mathbf{p}, s)\mathrm{d}s \\ y(\mathbf{p}, s_f) &= \int_0^s \sin\theta(\mathbf{p}, s)\mathrm{d}s \quad \kappa(\mathbf{p}, s_f) = u(\mathbf{p}, s). \end{aligned} \tag{58}$$

In the following, the notation for the parameter vector is:

$$\mathbf{p} = [a \quad b \quad c \quad d \ldots]^{\mathrm{T}}. \tag{59}$$

It will sometimes be convenient to define the augmented parameter vector which includes the final arc length as follows:

$$\mathbf{q} = [\mathbf{p}^T s_f]^{\mathrm{T}} = [a \quad b \quad c \quad d \ldots s_f]^{\mathrm{T}}. \tag{60}$$

Unless it is necessary, the general arc length will often not be distinguished from the terminal arc length s_f.

A.1. Partial Derivatives of Heading and Curvature

When curvature is given by a polynomial spiral, so is heading $\theta(\mathbf{p}, s)$:

$$\begin{aligned} \kappa(\mathbf{p}, s) &= a + bs + cs^2 + ds^3 + es^4 + \ldots \\ \theta(\mathbf{p}, s) &= as + b\frac{s^2}{2} + c\frac{s^3}{3} + d\frac{s^4}{4} + e\frac{s^5}{5} + \ldots . \end{aligned} \tag{61}$$

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

For notational convenience, we define the derivatives:

$$\begin{aligned}\kappa_f &= \kappa_0 + bs_f + cs_f^2 + ds_f^3 \\ \kappa_f' &= \left.\frac{\partial \kappa_f}{\partial s}\right|_{s=s_f} = b + 2cs_f + 3ds_f^2 \\ \kappa_f'' &= \left.\frac{\partial \kappa_f'}{\partial s}\right|_{s=s_f} = 2c + 6ds_f.\end{aligned} \tag{62}$$

The partial derivatives of these expressions with respect to the parameters and arc length are immediate:

$$\begin{aligned}\frac{\partial \theta}{\partial \mathbf{q}} &= \left[s_f \frac{s_f^2}{2} \frac{s_f^3}{3} \frac{s_f^4}{4} \ldots \kappa_f\right] \\ \frac{\partial \kappa}{\partial \mathbf{q}} &= \left[1 s_f s_f^2 s_f^3 \ldots \kappa_f'\right].\end{aligned} \tag{63}$$

The second derivative or Hessian matrix for heading is:

$$\frac{\partial^2 \theta}{\partial \mathbf{q}^2} = \begin{bmatrix} 0 & 0 & 0 & 0 & \ldots & 1 \\ 0 & 0 & 0 & 0 & \ldots & s_f \\ 0 & 0 & 0 & 0 & \ldots & s_f^2 \\ 0 & 0 & 0 & 0 & \ldots & s_f^3 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & s_f & s_f^2 & s_f^3 & \ldots & \kappa_f' \end{bmatrix}. \tag{64}$$

Similarly, the Hessian for curvature is:

$$\frac{\partial^2 \kappa}{\partial \mathbf{q}^2} = \begin{bmatrix} 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 0 & \ldots & 1 \\ 0 & 0 & 0 & 0 & \ldots & 2s_f \\ 0 & 0 & 0 & 0 & \ldots & 3s_f^2 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 1 & 2s_f & 3s_f^2 & \ldots & \kappa_f'' \end{bmatrix}. \tag{65}$$

A.2. Partial Derivatives of the Position Integrals

The position states are given by the integrals:

$$x(\mathbf{p}, s_f) = \int_0^{s_f} \cos\theta(\mathbf{p}, s)\mathrm{d}s \qquad y(\mathbf{p}, s_f) = \int_0^{s_f} \sin\theta(\mathbf{p}, s)\mathrm{d}s. \tag{66}$$

More generally, we define the two nth-order gradient integrals:

$$C^n(s) = \int_0^{s} s^n \cos\theta(s)\mathrm{d}s \qquad S^n(s) = \int_0^{s} s^n \sin\theta(s)\mathrm{d}s. \tag{67}$$

Clearly

$$x(s) = C^0(s) \qquad y(s) = S^0(s). \tag{68}$$

For polynomial spirals, partial derivatives of these integrals with respect to arc length and spiral parameters satisfy useful recurrence relationships. For example:

$$\begin{aligned}\frac{\partial}{\partial s}C^n(s) &= s^n\cos\theta(s) & \frac{\partial}{\partial s}S^n(s) &= s^n\sin\theta(s) \\ \frac{\partial}{\partial a}C^n(s) &= -S^{n+1}(s) & \frac{\partial}{\partial a}S^n(s) &= C^{n+1}(s) \\ \frac{\partial}{\partial b}C^n(s) &= -\frac{1}{2}S^{n+2}(s) & \frac{\partial}{\partial b}S^n(s) &= \frac{1}{2}C^{n+2}(s) \\ \frac{\partial}{\partial c}C^n(s) &= -\frac{1}{3}S^{n+3}(s) & \frac{\partial}{\partial c}S^n(s) &= \frac{1}{3}C^{n+3}(s) \\ &\ldots & &\ldots.\end{aligned} \tag{69}$$

The evaluation of these integrals represents the fundamental computational burden of this approach to trajectory generation. A subsequent section will describe a technique for their efficient evaluation.

The partial derivatives of the position integrals with respect to the parameters and arc length are related to higher-order gradient integrals

$$\begin{aligned}\frac{\partial x}{\partial \mathbf{q}} &= \left[-S^1(s_f) - \frac{1}{2}S^2(s_f) - \frac{1}{3}S^3(s_f) - \frac{1}{4}S^4(s_f) \ldots c\theta_f\right] \\ \frac{\partial y}{\partial \mathbf{q}} &= \left[C^1(s_f) - \frac{1}{2}C^2(s_f) - \frac{1}{3}C^3(s_f) - \frac{1}{4}C^4(s_f) \ldots s\theta_f\right]\end{aligned} \tag{70}$$

where $c\theta_f = \cos(\theta_f)$, etc. The Hessian matrix for x is

$$\frac{\partial^2 x}{\partial \mathbf{q}^2} = \tag{71}$$

$$\begin{bmatrix} -C^2(s_f) & -\frac{1}{2}C^3(s_f) & -\frac{1}{3}C^4(s_f) & -\frac{1}{4}C^5(s_f) & \ldots & -s_f s\theta_f \\ -\frac{1}{2}C^3(s_f) & -\frac{1}{4}C^4(s_f) & -\frac{1}{6}C^5(s_f) & -\frac{1}{8}C^6(s_f) & \ldots & -\frac{s_f^2}{2}s\theta_f \\ -\frac{1}{3}C^4(s_f) & -\frac{1}{6}C^5(s_f) & -\frac{1}{9}C^6(s_f) & -\frac{1}{12}C^7(s_f) & \ldots & -\frac{s_f^3}{3}s\theta_f \\ -\frac{1}{4}C^5(s_f) & -\frac{1}{8}C^6(s_f) & -\frac{1}{12}C^7(s_f) & -\frac{1}{16}C^8(s_f) & \ldots & -\frac{s_f^4}{4}s\theta_f \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ -s_f s\theta_f & -\frac{s_f^2}{2}s\theta_f & -\frac{s_f^3}{3}s\theta_f & -\frac{s_f^4}{4}s\theta_f & \ldots & -\kappa_f s\theta_f \end{bmatrix}.$$

The Hessian matrix for y is:

$$\frac{\partial^2 y}{\partial \mathbf{q}^2} = \tag{72}$$

$$\begin{bmatrix} -S^2(s_f) & -\frac{1}{2}S^3(s_f) & -\frac{1}{3}S^4(s_f) & -\frac{1}{4}S^5(s_f) & \ldots & -s_f c\theta_f \\ -\frac{1}{2}S^3(s_f) & -\frac{1}{4}S^4(s_f) & -\frac{1}{6}S^5(s_f) & -\frac{1}{8}S^6(s_f) & \ldots & -\frac{s_f^2}{2}c\theta_f \\ -\frac{1}{3}S^4(s_f) & -\frac{1}{6}S^5(s_f) & -\frac{1}{9}S^6(s_f) & -\frac{1}{12}S^7(s_f) & \ldots & -\frac{s_f^3}{3}c\theta_f \\ -\frac{1}{4}S^5(s_f) & -\frac{1}{8}S^6(s_f) & -\frac{1}{12}S^7(s_f) & -\frac{1}{16}S^8(s_f) & \ldots & -\frac{s_f^4}{4}c\theta_f \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ -s_f c\theta_f & -\frac{s_f^2}{2}c\theta_f & -\frac{s_f^3}{3}c\theta_f & -\frac{s_f^4}{4}c\theta_f & \ldots & -\kappa_f c\theta_f \end{bmatrix}.$$

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

A.3. Partial Derivatives for Smoothness

While many metrics for measuring path smoothness are possible, the following one will be illustrated here:

$$J_\kappa[\kappa(\mathbf{p}, s)] = \frac{1}{2}\int_0^{s_f} [\kappa(\mathbf{p}, s)]^2 \mathrm{d}s. \tag{73}$$

This functional will discourage sharp turns in favor of more graceful ones. It can be evaluated easily in closed form but it is convenient to do so numerically given all of the other integrals which must be evaluated numerically.

For polynomial spirals, its gradient is:

$$\begin{aligned} \frac{\partial J_\kappa}{\partial \mathbf{q}} &= \left[\frac{\partial J_\kappa}{\partial a}\frac{\partial J_\kappa}{\partial b}\frac{\partial J_x}{\partial c}\cdots\frac{\partial J_\kappa}{\partial s_f}\right] \\ \frac{\partial J_\kappa}{\partial \mathbf{q}} &= \int_0^{s_f} \kappa\frac{\partial \kappa}{\partial \mathbf{p}}\mathrm{d}s \qquad \frac{\partial J_\kappa}{\partial s_f} = \frac{1}{2}\kappa(\mathbf{p}, s_f)^2. \end{aligned} \tag{74}$$

In order to express the parameter gradients, it is useful to define the following general gradient integral:

$$K^n(s) = \int_0^s s^n \kappa(s)\mathrm{d}s. \tag{75}$$

For polynomial spirals, these integrals can be evaluated in closed form:

$$\begin{aligned} K^0(s) &= \theta(s) = as + \frac{b}{2}s^2 + \frac{c}{3}s^3 + \ldots \\ K^1(s) &= a\frac{s^2}{2} + b\frac{s^3}{3} + c\frac{s^4}{4} + \ldots \\ K^2(s) &= a\frac{s^3}{3} + b\frac{s^4}{4} + c\frac{s^5}{5} + \ldots \\ &\ldots \\ K^n(s) &= a\frac{s^{n+1}}{n+1} + b\frac{s^{n+2}}{n+2} + \ldots . \end{aligned} \tag{76}$$

The gradients of the smoothness performance index can now be written in terms of these gradient integrals:

$$\begin{aligned} \frac{\partial J_\kappa}{\partial \mathbf{p}} &= \int_0^{s_f} \kappa \begin{bmatrix} 1 \\ s \\ s^2 \\ \cdots \end{bmatrix}^{\mathrm{T}} \mathrm{d}s = \begin{bmatrix} K^0(s_f) \\ K^1(s_f) \\ K^2(s_f) \\ \cdots \end{bmatrix}^{\mathrm{T}} \frac{\partial J_\kappa}{\partial s_f} \\ &= \frac{1}{2}\kappa(\mathbf{p}, s_f)^2. \end{aligned} \tag{77}$$

The gradients of these integrals satisfy:

$$\begin{aligned} \frac{\partial}{\partial a}K^n(s) &= \frac{s^{n+1}}{n+1} \qquad \frac{\partial}{\partial s}K^n(s) = s^n\kappa(s) \\ \frac{\partial}{\partial b}K^n(s) &= \frac{s^{n+2}}{n+2} \\ \frac{\partial}{\partial c}K^n(s) &= \frac{s^{n+3}}{n+3} \\ &\ldots \quad . \end{aligned} \tag{78}$$

The Hessian matrix for the performance index is

$$\frac{\partial^2 J_\kappa}{\partial \mathbf{q}^2} = \begin{bmatrix} s_f & \frac{s_f^2}{2} & \frac{s_f^3}{3} & \ldots & \kappa_f \\ \frac{s_f^2}{2} & \frac{s_f^3}{3} & \frac{s_f^4}{4} & \ldots & s_f\kappa_f \\ \frac{s_f^3}{3} & \frac{s_f^4}{4} & \frac{s_f^5}{5} & \ldots & s_f^2\kappa_f \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \kappa_f & s_f\kappa_f & s_f^2\kappa_f & \ldots & \kappa_f\kappa_f' \end{bmatrix}. \tag{79}$$

Appendix B: Efficient Evaluation of Quadrature Gradients

Consider the problem of computing the gradients $\partial x/\partial \mathbf{p}$ and $\partial y/\partial \mathbf{p}$ and Hessians of the position integrals. This is equivalent to computing the quadratures:

$$C^n(s) = \int_0^s s^n \cos\theta(s)\mathrm{d}s \qquad S^n(s) = \int_0^s s^n \sin\theta(s)\mathrm{d}s. \tag{80}$$

Finite differences could be used to differentiate these integrals by evaluating and differencing these integrals for two nearby values of any parameter. However, it is less computationally intensive and potentially more accurate to apply quadrature to the exact derivative.

This can be illustrated with Simpson's rule. We define the notation for the integrands:

$$\begin{aligned} x(\mathbf{p}, s_f) &= \int_0^{s_f} \cos\theta(\mathbf{p}, s)\mathrm{d}s = \int_0^{s_f} f(\mathbf{p}, s)\mathrm{d}s \\ y(\mathbf{p}, s_f) &= \int_0^{s_f} \sin\theta(\mathbf{p}, s)\mathrm{d}s = \int_0^{s_f} g(\mathbf{p}, s)\mathrm{d}s. \end{aligned} \tag{81}$$

Under Simpson's rule, the interval $[0, s]$ is divided into n equal segments of width Δs (where n is even). At each of the resulting $n+1$ values of s, the integrands and $f()$ and $g()$ are evaluated. Then, the integrals are approximated by:

$$\begin{aligned} x(\mathbf{p}) &= \frac{\Delta s}{3}\{f_0 + 4f_1 + 2f_2 + \ldots + 2f_{n-2} \\ &\quad + 4f_{n-1} + f_n\} \\ y(\mathbf{p}) &= \frac{\Delta s}{3}\{g_0 + 4g_1 + 2g_2 + \ldots + 2g_{n-2} \\ &\quad + 4g_{n-1} + g_n\}. \end{aligned} \tag{82}$$

Now the derivatives with respect to arc length are the integrands themselves. At any point s_k:

$$\left.\frac{\partial}{\partial s}x(\mathbf{p}, s)\right|_{s=s_k} = f_k \qquad \left.\frac{\partial}{\partial s}y(\mathbf{p}, s)\right|_{s=s_k} = g_k. \tag{83}$$

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

On the other hand, the derivatives with respect to the parameters follow a pattern similar to the following partial derivative for the a parameter. At any point s_k:

$$\begin{aligned}\frac{\partial f_k}{\partial a} &= \frac{\partial}{\partial a}\cos[\theta(\mathbf{p})] = -\sin[\theta(\mathbf{p})]\frac{\partial \theta}{\partial a} = -s_k g_k \\ \frac{\partial g_k}{\partial a} &= \frac{\partial}{\partial a}\sin[\theta(\mathbf{p})] = \cos[\theta(\mathbf{p})]\frac{\partial \theta}{\partial a} = s_k f_k. \end{aligned} \quad (84)$$

The partials of the integrals evaluated at the endpoints are then

$$\begin{aligned}&\frac{\partial}{\partial a}x(\mathbf{p}) = \\ &\frac{\Delta s}{3}\left\{\frac{\partial f_0}{\partial a} + 4\frac{\partial f_1}{\partial a} + 2\frac{\partial f_2}{\partial a} + \ldots + 4\frac{\partial f_{n-1}}{\partial a} + \frac{\partial f_n}{\partial a}\right\} \\ &\frac{\partial}{\partial a}y(\mathbf{p}) = \\ &\frac{\Delta s}{3}\left\{\frac{\partial g_0}{\partial a} + 4\frac{\partial g_1}{\partial a} + 2\frac{\partial g_2}{\partial a} + \ldots + 4\frac{\partial g_{n-1}}{\partial a} + \frac{\partial g_n}{\partial a}\right\}. \end{aligned} \quad (85)$$

Substituting the results for the partials gives

$$\begin{aligned}\frac{\partial}{\partial a}x(\mathbf{p}) &= -\frac{\Delta s}{3}\{s_0 g_0 + 4s_1 g_1 + 2s_2 g_2 + \ldots + 2s_{n-2}g_{n-2} \\ &\quad +4s_{n-1}g_{n-1} + s_n g_n\} \\ \frac{\partial}{\partial a}y(\mathbf{p}) &= \frac{\Delta s}{3}\{s_0 f_0 + 4s_1 f_1 + 2s_2 f_2 + \ldots + 2s_{n-2}f_{n-2} \\ &\quad +4s_{n-1}f_{n-1} + s_n f_n\}. \end{aligned} \quad (86)$$

This is the same result that would be obtained by applying Simpson's rule to the closed form derivatives in eq. (69).

We define the weight vector:

$$\mathbf{w} = \{w_k\} = [1 \quad 4 \quad 2 \quad 4 \ldots 4 \quad 2 \quad 4 \quad 1]^{\mathrm{T}}. \quad (87)$$

Simpsons's rule is

$$x(\mathbf{p}) = \left(\frac{\Delta s}{3}\right)\sum_k w_k f_k \quad y(\mathbf{p}) = \left(\frac{\Delta s}{3}\right)\sum_k w_k g_k. \quad (88)$$

More generally, we define the sums approximating the integrals of eqs. (67) and (80):

$$\begin{aligned}C^n(s_k) \approx F_k^n &= \left(\frac{\Delta s}{3}\right)\sum_k s_k^n w_k f_k \\ S^n(s_k) \approx G_k^n &= \left(\frac{\Delta s}{3}\right)\sum_k s_k^n w_k g_k. \end{aligned} \quad (89)$$

Clearly

$$F_k^0 \approx x(s_k) \qquad G_k^0 \approx y(s_k). \quad (90)$$

The partial derivatives of the quadratures follow analogous recurrence relationships:

$$\begin{aligned}\frac{\partial F_k^n}{\partial s_k} &= s_k^n f_k & \frac{\partial G_k^n}{\partial s_k} &= s_k^n g_k \\ \frac{\partial F_k^n}{\partial a} &= -G^{(n+1)}(k) & \frac{\partial G_k^n}{\partial a} &= F^{(n+1)}(k) \\ \frac{\partial F_k^n}{\partial b} &= -\frac{1}{2}G^{(n+2)}(k) & \frac{\partial G_k^n}{\partial b} &= \frac{1}{2}F^{(n+2)}(k) \\ \frac{\partial F_k^n}{\partial c} &= -\frac{1}{3}G^{(n+3)}(k) & \frac{\partial G_k^n}{\partial c} &= \frac{1}{3}F^{(n+3)}(k) \\ &\ldots & &\ldots . \end{aligned} \quad (91)$$

Any order of derivative can be computed by accumulating the sums F_k^n and G_k^n up to the appropriate order. Conceptually, the computation proceeds by filling in a structure similar to Table 1 one column at a time. The case for integrals up to order 3 is illustrated. Columns can be lengthened progressively by doubling their length (and halving Δs) until all of the integrals being computed pass a convergence test. As long as the weights w_k are computed for each row, there is no need to order the columns to correspond to increasing arc length.

It is possible with little effort to combine each previous iteration with the next one without recomputing the sums of the columns all over again. Each term weighted by 4 (k odd) in a given iteration is weighted by 2 in the next (and all subsequent iterations) whereas the terms weight by 1 or 2 (k even) have their weights unchanged. Accordingly, if the sums of odd and even terms are computed separately, they are easily converted to their contribution to the next iteration.

Appendix C: Exact Solution to the Linear Constraints

The constraint equations are again of the form:

$$\begin{aligned}\kappa(s) &= a + bs + cs^2 + ds^3 + \ldots \\ \theta(s) &= as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \ldots \\ x(s) &= \int_0^s \cos\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \ldots\right]\mathrm{d}s \\ y(s) &= \int_0^s \sin\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \ldots\right]\mathrm{d}s. \end{aligned} \quad (92)$$

We consider the five-parameter fully-constrained curve; for example purposes, the constraint equations take the form:

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

Table 1. Computation of Position Quadrature Gradients and Hessians

k	w_k	θ_k	f_k	g_k	s_k	s_k^2	s_k^3	$w_k f_k$	$w_k g_k$	$w_k s_f^2 f_k$	$w_k s_f^2 g_k$	$w_k s_f^3 f_k$	$w_k s_f^3 g_k$
0	1	θ_0	f_0	g_0	s_0	s_0^2	s_0^3	$w_0 f_0$	$w_0 g_0$	$w_0 s_0^2 f_0$	$w_0 s_0^2 g_0$	$w_0 s_0^3 f_0$	$w_0 s_0^3 g_0$
1	4	θ_1	f_1	g_1	s_1	s_1^2	s_1^3	$w_1 f_1$	$w_1 g_1$	$w_1 s_1^2 f_1$	$w_1 s_1^2 g_1$	$w_1 s_1^3 f_1$	$w_1 s_1^3 g_1$
...	...	...	...	...	...	...	...	...	...	...	...	...	...
n	1	θ_n	f_n	g_n	s_n	s_n^2	s_n^3	$w_n f_n$	$w_n g_n$	$w_n s_n^2 f_n$	$w_n s_n^2 g_n$	$w_n s_n^3 f_n$	$w_n s_n^3 g_n$
								x_n	y_n	F_n^2	G_n^2	F_n^3	G_n^3

$$
\begin{aligned}
\kappa 0 &= a \\
\kappa_f &= a + bs_f + cs_f^2 + ds_f^3 \\
\theta_f &= as_f + \frac{bs_f^2}{2} + \frac{cs_f^3}{3} + \frac{ds_f^4}{4} \\
x_f &= \int_0^{s_f} \cos\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \ldots\right] \mathrm{d}s \\
y_f &= \int_0^{s_f} \sin\left[as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \ldots\right] \mathrm{d}s. \qquad (93)
\end{aligned}
$$

It is important to recognize that all but two of these equations are linear in all parameters but s_f. This creates an opportunity to guarantee that all but two constraints are perfectly satisfied in each iteration of the algorithm.

Fix the values of s_f and one other parameter, say d, and solve the trivial first equation, then the second and third equations can be written as:

$$
\begin{bmatrix} s_f & s_f^2 \\ \frac{s_f^2}{2} & \frac{s_f^3}{3} \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} \kappa_f - \kappa 0 - ds_f^3 \\ \theta_f - \kappa 0 s_f - \frac{ds_f^4}{4} \end{bmatrix} \qquad (94)
$$

or, if b is chosen to be dependent,

$$
\begin{bmatrix} s_f^2 & s_f^3 \\ \frac{s_f^3}{3} & \frac{s_f^4}{4} \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \kappa_f - \kappa 0 - bs_f \\ \theta_f - \kappa 0 s_f - \frac{bs_f^2}{2}. \end{bmatrix}
$$

This set is easily solvable for b and c (or c and d). Note particularly that it cannot be singular for curves of nonzero length. If these equations are used, the final heading and final curvature are guaranteed to be correct at the end of each iteration. The iterations then become confined to the problem of moving the endpoint to the right place.

References

Boisonnat, J., Cérézo, A., and Leblond, J. 1992. Shortest paths of bounded curvature in the plane. In *Proceedings of the ICRA*, Nice, France, pp. 2315–2320.

Brockett, R. W. 1981. Control theory and singular Riemannian geometry. In *New Directions in Applied Mathematics*, Springer-Verlag, New York, pp. 11–27.

Dillen, F. 1990. The classification of hyper-surfaces of a Euclidean space with parallel higher fundamental form. *Math. Z.* 203:635–643.

Delingette, H., Herbert, M., and Ikeuchi, K. 1991. Trajectory generation with curvature constraint based on energy minimization. In *Proceedings of the IROS*, Osaka, Japan, pp. 206–211.

Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79:497–516.

Fernandes, C., Gurvits, L., and Li, Z. X. 1991. A variational approach to optimal nonholonomic motion planning. *IEEE International Conference on Robotics and Automation*, Sacramento, CA, pp. 680–685.

Horn, B. K. 1983. The curve of least energy. *ACM Transactions on Mathematical Software* 9(4):441–460.

Kanayama, Y., and Miyake, N. 1985. Trajectory generation for mobile robots. *Robotics Research*, MIT Press, Cambridge, MA.

Kanayama, Y., and Hartman, B. I. 1988. Smooth local path planning for autonomous vehicles, Technical Report, Department of Computer Science, University of California, Santa Barbara, CA.

Kano, H., Egerstedt, M., Nakata, H., and Martin, C.F. 2003. B-splines and control theory. *Applied Mathematics and Computation*, to appear.

Kelly, A. November 2001. General solution for linearized error propagation in vehicle odometry. In *International Symposium on Robotics Research*, Lorne, Victoria, Australia.

Komoriya, K., Tachi, S., and Tanie, K. 1984. A method for autonomous locomotion of mobile robots. *Journal of the Robotics Society of Japan* 2:222–231.

Kuhn, H. W., and Tucker, A. W. 1961. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, J. Keyman ed., University of California Press, Berkeley, CA, pp. 481–492.

Latombe, J. C. 1991. *Robot Motion Planning*, Kluwer Academic, Dordrecht.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009

Laumond, J. P., ed. 1998. Robot motion planning and control, LAAS report 97438.

Luenberger, D. G. 1989. *Linear and Nonlinear Programming*, 2nd edition, Addison Wesley, Reading, MA.

Murray, R., and Sastry, S. 1993. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control* 38(5).

Nagy, B., and Kelly, A. 2001. Trajectory generation for car-like robots using cubic curvature polynomials. In *Field and Service Robots*, 11 June, Helsinki, Finland.

Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. 1988. *Numerical Recipes in C*, Cambridge University Press, Cambridge.

Reeds, J. A., and Shepp, R. A. 1990. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics* 145(2).

Reuter, J. October 1998. Mobile robot trajectories with continuously differentiable curvature: an optimal control approach. In *Proceedings of the 1998 IEEE/RSJ Conference on Intelligent Robots and Systems*, Victoria, BC, Canada.

Shin, D. H., and Singh, S. 1990. Path generation for robot vehicles using composite clothoid segments, Technical Report, CMU-RI-TR- 90-31, The Robotics Institute, Carnegie Mellon University.

Shkel, A., and Lumelsky, V. 1996. On calculation of optimal paths with constrained curvature: the case of long paths.

Slama, C. C., Theurer, C., and Henriksen, S. W., eds. 1980. *Manual of Photogrammetry*, American Society of Photogrammetry.

Tilbury, D., Murray, R., and Sastry, S. 1995. Trajectory Generation for the N Trailer Problem using the Goursat Normal Form. *IEEE Transactions on Automatic Control* 40(5):802–819.

Tsumura, T., Fujiwara, N., Shirakawa, T., and Hashimoto, M. October 1981. An Experimental System for Automatic Guidance of a Robotic Vehicle Following a Route Stored in Memory. In *Proceedings of the 11th International Symposium on Industrial Robots*, pp. 187–193.

Downloaded from http://ijr.sagepub.com at University of Sydney on August 9, 2009