in Field and Service Robots 2001
Helsinki, Finland, June 11, 2001

TRAJECTORY GENERATION FOR CAR-LIKE ROBOTS USING CUBIC CURVATURE POLYNOMIALS

Bryan Nagy and Alonzo Kelly

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890
email: bnagy@rec.ri.cmu.edu, alonzo@ri.cmu.edu
url: http://www.ri.cmu.edu/people/nagy_bryan_ext.html, http://www.frc.ri.cmu.edu/~alonzo

Abstract: Curvature polynomials of cubic order are ideal primitive trajectories for car-like robots. Unlike the clothoids, which are linear curvature polynomials, cubic curves can be used to determine a unique trajectory to an arbitrary target posture using a single continuous primitive. Such curves are also the lowest order curves which are continuous in the torque applied to steering mechanisms, so they generate trajectories which are relatively easily tracked by a real vehicle. Like the clothoids, cubic curvature polynomials are relatively difficult to compute but are easy to execute. A real-time numerical method to compute them is described.

Keywords: mobile robots, car-like robots, trajectory generation, curve generation, nonholonomic

1 INTRODUCTION

The fields of manipulators and mechanisms have enjoyed the ability to compute inverse kinematics for some time [Denavitt and Hartenberg, 1955]. The equivalent problem for car-like robots is that of determining the control input which causes the vehicle to achieve a goal position (x, y), pose (x, y, θ) or posture (x, y, θ, κ). Earlier literature has sometimes used the word posture to mean the 3-tuple which is generally known as pose today. We will use posture to refer to the 4-tuple which includes curvature.

While vehicles are typically actuated in rate rather than position, this does not change the fact that there are many situations where it is necessary to achieve a precise position and the relevant inputs must be found to accomplish the required motion. This paper discusses a method that we have been using for the control of various automated guided vehicles which are required to achieve precise postures and follow precise trajectories.

1.1 Setup

The mobile robot "inverse kinematics" problem is considerably more difficult than the manipulator problem, in part because it is not a kinematics problem - it is a kinetics problem. One simplified variant of the relevant state equations is:

$$\begin{aligned} \dot{x} &= V(t)\cos\theta(t) \\ \dot{y} &= V(t)\sin\theta(t) \\ \dot{\theta} &= \kappa(t)V(t) \\ \dot{\kappa} &= \dot{\alpha}(t)/L \end{aligned} \tag{1}$$

The equations of a real system would require the addition of terms to model the effect of vehicle mass but these terms do not affect the subject matter of the paper. The important point is that we have two inputs that control four outputs.

We consider the curvature a state because the steering mechanism has mass and is subject to restoring forces that depend on vehicle state and terrain. These steering dynamics issues mean that the steering actuator will move continuously and that controllers which ask it to move discontinuously will incur following error that may be unnecessary.

The linear velocity V and steer angle velocity $\dot{\alpha}$ are the control inputs. The wheelbase is L and the state vector is the posture defined above.

The equivalent of inverse kinematics for a manipulator is the problem of determining the control signal over some period of time which will cause the vehicle to achieve a goal posture. In manipulation, we solve kinematic equations for a point in configuration space. In mobile robots, we must solve an underdetermined differential equation for a vector trajectory which achieves some unknown state trajectory that ends at the goal posture.

1.2 Motivation

This paper introduces methods to compute control inputs for an Ackerman steer or other car-like robot described by the state equations above. The velocity input does not affect trajectory shape so it will be ignored in the balance of the paper and we will switch to a specification of trajectories in terms of distance rather than time.

Consider the solution to the state equations for a curvature input which is a third order polynomial in arc length:

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3 \tag{2}$$

$$\theta(s) = \theta_0 + \int_0^S \kappa(s) = \theta_0 + \kappa_0 s + \frac{as^2}{2} + \frac{bs^3}{3} + \frac{cs^4}{4}$$

$$x(s) = x_0 + \int_0^S \cos(\theta(s)) \qquad y(s) = y_0 + \int_0^S \sin(\theta(s))$$

We will refer to these control inputs as cubics in the balance of the paper. This type of control input specification has many advantages for our target application.

1.2.1 Minimum Complexity / Sufficient Variability

Note that there are four parameters $p = [a, b, c, s]^T$ which determine the values of four states $x = [x, y, \theta, \kappa]^T$. The initial position and heading can be used to re-express the goal posture in the body frame and the initial curvature is included. Cubics are therefore the minimum order of polynomial which possesses sufficient degrees of freedom to achieve an arbitrary endpoint posture with a single primitive curve. Cubics can go from anywhere to anywhere subject only to the intrinsic limitations of the vehicle.

1.2.2 Feasibility

Such inputs are continuous in the third derivative of steering angle and hence are smooth in the torque applied to the steering actuator. This property makes them intrinsically executable subject only to kinematic limitations on the steering actuator and power limitations on the ability to accelerate it fast enough. There is usually no need to post-process or 'smooth' such inputs to make them match the capabilities of a vehicle. Unlike the case for constant curvature arcs and clothoids, there is no need to compute large numbers of them to approximate some other curve which is more easily executed.

1.2.3 Span the Set Of Feasible Controls

Cubics cover the set of all curves for which the fourth and all higher derivatives of curvature vanish:

$$\frac{\partial^{\nu}\kappa}{\partial s^{\nu}} = 0 \qquad \nu \geq 4 \tag{3}$$

Higher derivatives of curvature imply higher frequency inputs which will be progressively more attenuated by the dynamics. Hence, cubics span the space of all feasible controls up to some limit of frequency content.

To the degree that any optimal controller searches and ranks all possible controls, cubics can therefore approximate the optimal control signal very well and hence near optimal performance should be achieved.

1.2.4 Efficient Representation

While it will turn out to be difficult to compute the coefficients of the polynomial, cubics are convenient specifications of control signals. They are easily evaluated at any particular arc length in 8 floating point operations.

1.2.5 Compact Representation

Cubics are also a compact trajectory description - requiring only 5 numbers. They can be used to send lower-frequency updates over lower bandwidth connections. A complicated trajectory to move from any posture to any other posture can be downloaded and the controller can be expected to achieve the goal with no further intervention from higher level processes.

Many hierarchical control systems which update commands to lower levels at high rates do so simply to approximate a higher frequency curve with a sequence of lower frequency ones. If the controller interface is defined in terms of steer angle or curvature, or some other expression equivalent to a constant radius arc, higher level processes are potentially being robbed of resources just to interpolate signals.

Cubics eliminate the high frequency I/O which is often generated over the controller interface in order to approximate the steering control signal with piecewise constant radius arcs. This property tends to place more of the servo computation lower in the control hierarchy where loops are normally closed faster, so it generates less error due to time discretization. However, this lower update rate comes at the cost of reasoning at higher levels with more complicated trajectories.

1.2.6 Finer Control over Trajectories

Cubics control final heading *and* final curvature and completely determine the trajectory to the goal. This representation permits the measurement of servo errors around a nominally feasible trajectory throughout the path to the endpoint. Unlike arcs, for example, the inherent feasibility of the cubic trajectory allows control algorithms to eliminate endpoint errors in curvature and heading while there is still time to do so.

1.3 Prior Work

Kanayama's original work on clothoids introduced the idea of using continuous piecewise linear curvature curves for robot trajectory generation. This was novel because of its concern with producing posture continuous (instead of merely pose continuous) trajectories. However, as good as piecewise clothoids are, they still result in a discontinuity in the derivative of curvature.

Kanayama recognized the importance of curvature continuity for curves to be tracked by a PWS vehicle in [Kanayama and Nilipour, 1988] where he proposed a classification of curves based on their continuity in space, heading and curvature. In [Kanayama and Hartman, 1988] he dealt with the issue of smoothness of curves specifically for ackerman-steered vehicles. In this context he developed

"cubic spirals" - cubic polynomials in distance to represent the heading state trajectory. This was done by optimizing a cost function intended to maximize passenger comfort.

Delingette et. al. [1991] developed a family of trajectories called "intrinsic splines". This family includes our concept of cubic curvature polynomial under the designation IS3. These authors defined the order of postures based on the order of the highest derivative of heading in the configuration. They use a method which progressively deforms a straight line between the start and end postures, generating a discrete trajectory, using a cost function and deformation functional. Coefficients are then extracted from the converged result. To achieve compatibility with the maximum curvature constraints of a vehicle, they split the curve into splines at points of maximum curvature, moving those points in the direction perpendicular to the heading of the curve at that point. At the time of publication, the level of computation involved was considered to require parallel processing.

Our approach differs from earlier work in that we consider constraints of higher order than Kanayama does. Our considerations of smoothness of steer angle acceleration turn out to be more stringent than the ones proposed by many earlier authors for carrying passengers. Our higher order primitive has several other advantages. It converts an underconstrained problem requiring a sequence (spline) of primitives into one which is fully constrained - for which a unique or slightly redundant solution exists.

In comparison to Delingette et. al, our approach achieves real-time trajectory generation by simply solving the integro-differential state equations in reverse by converting them into four simultaneous nonlinear equations for the endpoint posture expressed in terms of four unknown constant parameters.

2 FORMULATION

The four equations of (2) can be interpreted as a set of four nonlinear equations:

$$\begin{aligned} x &= f_1(a, b, c, s) \\ y &= f_2(a, b, c, s) \\ \theta &= f_3(a, b, c, s) \\ \kappa &= f_4(a, b, c, s) \end{aligned} \tag{4}$$

where the line integral notation has been suppressed for clarity. Let the subscript 0 denote the initial posture while an *f* subscript will denote the final posture. We will express all coordinates in terms of the initial posture so that the initial position and heading coordinates are zero. In this case, the constraints that must be satisfied are:

$$\begin{aligned} x(s_f) &= x_f & \kappa(0) &= \kappa_0 \\ y(s_f) &= y_f & \theta(s_f) &= \theta_f \\ & & \kappa(s_f) &= \kappa_f \end{aligned} \tag{5}$$

Note that s_f is itself a free variable. We will satisfy the initial curvature intrinsically and then consider the cubic curvature polynomial that solves the remaining four constraints to be of the form:

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3$$

Figure 1 shows a typical member of this family of curves. The four degrees of parameter freedom in

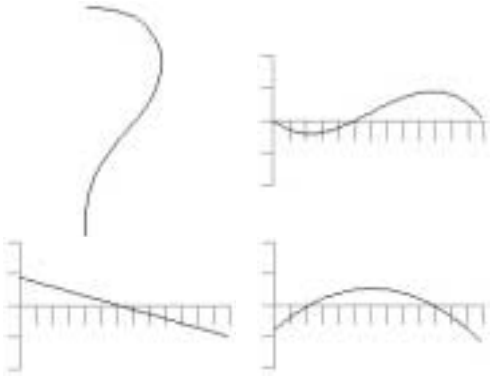

Figure 1: Clockwise from top left, a curve of cubic polynomial curvature, its curvature, and the first and second derivatives of curvature.

the cubic polynomial equations $\underline{p} = [a, b, c, s]^T$ are sufficient to generate the four degrees of freedom in the state space $\underline{x} = [x, y, \theta, \kappa]^T$. We can write equation (4) in vector form:

$$\underline{x} = \underline{f}(\underline{p}) \tag{6}$$

We can linearize the above system as follows:

$$\Delta\underline{x} = \left[\frac{\partial}{\partial \underline{p}}\underline{f}\right]\Delta\underline{p} \tag{7}$$

Sufficient degrees of freedom to control the entire endpoint state implies that the Jacobian has sufficient rank to invert, so that we can, in principle, solve for the parameters iteratively using:

$$\Delta\underline{p} = \left[\frac{\partial}{\partial \underline{p}}\underline{f}\right]^{-1}\Delta\underline{x} \tag{8}$$

Many previous publications have made the problem of curve generation appear very difficult in theory. We have found the above technique to be robust, easy to understand, and efficient in real applications.

3 IMPLEMENTATION

3.1 Initiation/Termination

The initial heuristics used to seed the iterative computation come from assuming the c parameter to be zero, using a heuristic to compute s, and then solving the $\theta(s)$ and $\kappa(s)$ equations simultaneously for a and b. The heuristic for s is based on an approximation of the observed average relationship between s and the total change in heading between the start and end postures over a large sample set. The actual equations are as follows:

$$d = \sqrt{x_f^2 + y_f^2} \qquad \Delta\theta = |\theta_f| \tag{9}$$

$$s = d\left(\frac{\Delta\theta^2}{5} + 1\right) + \frac{2}{5}\Delta\theta \qquad c = 0$$

$$a = \frac{6\theta_f}{s^2} - \frac{2\kappa_0}{s} + \frac{4\kappa_f}{s}$$

$$b = \frac{3}{s^2}(\kappa_0 + \kappa_f) + \frac{6\theta_f}{s^3}$$

In tracking, search, or replanning applications, the algorithm can be sped up drastically by reusing the parameters (a, b, c, s) from each cycle to seed the initial guess for the next cycle. This will usually result in much faster convergence, assuming the old and new end postures are in the same neighborhood.

The termination conditions for our algorithm were based on the accuracy in the destination position that was needed. We used the following values:

Table 1: Termination Conditions

Condition	Value
Allowable cross-track error	0.001 m
Allowable in-line error	0.001 m
Allowable heading error	0.1 rad
Allowable curvature error	0.005 1/m

3.2 Forward/Inverse Solution

The forward solution produces the posture (x, y, θ, κ) at any point along the curve, given the start posture, the parameters (a, b, c) and the distance traveled along the curve.

The inverse solution takes two postures as input, and produces the parameters that define a valid trajectory between them. The two postures are converted into a single relative posture to make the algorithm easier to implement and more generic.

3.3 Linearization

There is no closed form solution for the forward equations since they are related to the well-known, and essentially transcendental Fresnel integrals. So, for the forward solution we use numerical quadrature techniques to integrate the equations. For the inverse solution, since inverting the exact equations is apparently too difficult to perform, we use a finite difference approximation to the Jacobian and then invert it at each step in the iterative update algorithm sketched in equation (8).

Hence, the inverse solution is performed by successively performing the relatively straightforward forward solution. Starting from the heuristic guess parameters of equations (9), we generate the forward solution using numerical quadrature. The result is checked against the desired end posture. If it is within the termination conditions described above, then the algorithm terminates. Otherwise, we compute an approximate Jacobian by solving for a very nearby end posture and performing a matrix finite difference:

$$\frac{\partial f_{ij}}{\partial p_k} \approx \frac{[f_{ij}(p_k) - f_{ij}(p_k + h)]}{h} \qquad h = small$$

The difference between the desired and the present state vector is then multiplied by the inverted Jacobian to compute the necessary change to the parameter vector. The changes are scaled down to preserve stability, and then added to the present parameter vector. The algorithm then repeats with the forward solution and testing, continuing until either the termination conditions are met, or the maximum number of cycles is reached.

4 RESULTS/CONCLUSIONS

We have found the above algorithm to be eminently practical for our vehicle over the following relative end posture envelope:

$$-1.0m < x_f < 1.0m \qquad -\frac{4\pi}{5} < \theta_f < \frac{4\pi}{5}$$

$$1.0m < y_f < 5.0m \qquad -0.1\frac{1}{m} < \kappa_f < 0.1\frac{1}{m}$$

Figure 2 depicts some example curves:

Figure 2: Sample cubics with differing end postures

Average computation times over this envelope are summarized in the following table. The default heuristic case applies when equations (9) are used, whereas the previous parameters solution applies

when a nearby solution is already available.

Table 2: Runtimes

Initial Parameter Source	Avg. Runtime (sec.)
Default Heuristics	0.0134
Previous Cubic Parameters	0.00557

These times are well within the ranges necessary to facilitate real time operation of a robot vehicle. Note that, in general, runtimes are greater for curves that have larger absolute values of the c parameter.

4.1 Applications

There are a wide variety of applications to which cubics have been applied. The simplest is the use of cubics as the last fundamental representation of a vehicle motion command in the hierarchy above the individual axis servos. A low-level drive interface receives commands in the form of cubics (or any degenerate form thereof: lines, arcs, or clothoids) and executes steering servo commands based on the direct feed of distance traveled from the encoders.

Another basic use of cubics is the generation of a trajectory from a start to end posture which is then given to a lower level path tracker. By using cubics with a destination curvature of zero for this sort of motion, the vehicle achieves a much more straight-on approach to its destination, resulting in less angular or cross-track error as a function of error in distance traveled.

Splines of individual such cubics can be used to reach fixed points when operating in the proximity of obstacles. Individual trajectories can also be modified using equation (8) to move a point away from a near or predicted collision.

Cubics can be used to replace lines and arcs in AGV guidepath networks, producing a more compact representation because fewer primitives are needed. Also, since there are essentially no executable curves that cannot be represented by a cubic, all possible trajectories can be expressed in the network.

Independent of path planning and generation, cubics have also been used in our path tracking application. Here a cubic polynomial is generated from the vehicle's present posture to a goal posture at some look-ahead point on the planned path. This cubic is then passed to the lower level drive control, and is updated at each cycle of path tracking. It should also be noted that in this application especially, the reuse of the previous cubic parameters as the initial guess for each successive tracking cycle speeds the calculations up tremendously. The potential use of a goal curvature in this application can lead to much higher path following performance.

A more novel application of cubics is that of extremely smooth end posture servoing. There are situations in which a vehicle's desired end posture may change unpredictably based on sensor input. Replanning static cubic trajectories from each new (potentially erroneous) destination posture could result in the vehicle being unable to reach the destination by the time it knows where the destination actually is. In order to drive as conservatively as possible and maintain the direction of our information-providing sensors toward our target, we employ the following steering algorithm:

For any given tracking cycle we search over a space of cubics, modifying our theoretical present curvature κ_0, looking for the cuboid with the smallest absolute c parameter which reaches the present destination. The steer angle corresponding to that initial curvature is then sent as a command to the drive interface.

Since the function of κ_0 versus c in that space is smooth, with only one minimum in the neighborhood of 'reasonable' cubics from that start to end posture, simply commanding that corresponding steer angle guarantees stability. This results in a responsive and smooth approach to our destination.

Cubics can also be used as a means of testing kinematic feasibility of a goal posture. After generating a cuboid to the goal, its curvature and derivatives of curvature can be examined to determine if their extremes lie outside the performance envelope of the vehicle.

5 FUTURE WORK

5.1 Near Closed Form Solutions and Speed Ups

Clothoids can be solved using Fresnel Integrals. There are good approximations for Fresnel Integrals. Cubics can be expressed in terms of Fresnel integrals, so the forward solution can potentially be sped up and have its accuracy improved over the current algorithm.

Infinite series or asymptotic approximations to the forward solution promise more accurate and faster computation. In some cases, a good series approximation might be expected to convert the inverse problem into an approximating polynomial evaluation as well.

5.2 Lookup Tables

Extensive computing and curve-fitting to the surface of the space of cubics might yield a smaller lookup table-based representation which could be computed once and yield near-instantaneous results

thereafter. A classic lookup table, however, without functions approximating its shape, would take up far too much space (necessarily four dimensional) to be practical for high resolution results. It would be necessary to store the majority of the table in the form of approximating equations and their parameters, and from these derive extremely good starting points for future computations.

5.3 Existence/Uniqueness

There is no consideration of the limits of the vehicle motion inherent in the formulation, and because not all points are reachable directly (in less than 180 degrees of total rotation) by a vehicle, no guarantee of the existence of a *practical* solution exists. It may be useful to explore ways to integrate the limits on vehicle motion with the polynomial equations themselves.

It is easily demonstrated that there are at least two valid cubics for any given nondegenerate pair in a large set of start and end postures. For example, consider two start and end postures with curvature of zero, and headings that mutually point at each other as shown in Figure 3.

Figure 3: Two different (symmetrical) cubics reaching the same relative destination posture.

It is clear that a curve between these postures can either curve to one side or the other to get there. Once those two possible cubics are both computed, modifying the destination posture slightly and reusing the initial parameters of each cuboid produces two new (now non-symmetrical) cubics which both still reach the same relative posture.

Figure 4: Two different (non-symmetrical) cubics reaching the same relative destination posture.

More work is needed to examine all possible solutions and address the numerical instability which is expected to arise when two solutions converge.

5.4 Cost Computation/Optimality

In many practical problems, the end posture is specified inexactly and the permitted variation can be used to generate a space of possibilities over which to optimize. Eventually, we might like to compute the ‘best’ cubic curvature polynomial for any given application

6 REFERENCES

[Denavitt and Hartenberg, 1955] J Denavitt and R. S. Hartenberg, “A Kinematic Notation for Lower-Pair Mechanisms based on Matrices”, ASME Journal of Applied Mechanics, June 1955.

[Kanayama and Miyake, 1985] Y. Kanayama, N.Miyake, "Trajectory Generation for Mobile Robots", Artificial Intelligence Laboratory, Dept. of Computer Science, Stanford University, Stanford, CA, 1985.

[Kanayama, 1985] Y. Kanayama, "A Vehicle Control Architecture - SMOOTH DRIVER", Artificial Intelligence Laboratory, Stanford University, Stanford, CA, 1985.

[Shin and Singh, 1990] D. H. Shin and S. Singh, "Path Generation for Robot Vehicles Using Composite Clothoid Segments", Technical Report, CMU-RI-TR-90-31, The Robotics Institute, Carnegie Mellon University, 1990.

[Kanayama and Nilipour, 1988] Y. Kanayama, A. Nilipour, C.A. Lelm, "A Locomotion Control Method for Autonomous Vehicles", Technical Report, Dept. of Computer Science, University of California, Santa Barbara, 1988

[Kanayama and Hartman, 1988] Y. Kanayama and B.I. Hartman, "Smooth Local Path Planning for Autonomous Vehicles" Technical Report, Dept. of Computer Science, University of California, Santa Barbara, 1988.

[Delingette et al., 1991] H Delingette, M. Herbert, and K Ikeuchi, "Trajectory Generation with Curvature Constraint based on Energy Minimization", Proc, IROS, pp 206-211, Osaka, Japan, 1991.