

Efficient Cartesian Path Approximation for Robots Using Trigonometric Splines

Dan Simon
TRW, SB2-1051
PO Box 1310
San Bernardino, CA 92402
dsimon@thor.bmd.trw.com

Can Isik
Syracuse University
Dept. of Electrical Engineering
Syracuse, NY 13244

ABSTRACT

A smooth approximation of a desired robot path can be realized by interpolating a sequence of joint-space knots with a trigonometric spline. In this paper we derive the computational effort required for the formulation of trigonometric splines and show how real-time obstacle avoidance can be implemented. The required computational expense is calculated and compared to that of algebraic splines. In addition, we demonstrate analytically that the Cartesian path error resulting from the use of trigonometric splines is inversely proportional to the number of knots if certain assumptions are satisfied. We then verify this result numerically, and extend the result numerically to cases where the given assumptions are not satisfied.

I. INTRODUCTION

The industrial robot is a highly nonlinear, coupled multivariable system with nonlinear constraints. For this reason, robot control algorithms are often divided into two stages: *path planning* and *path tracking*. A conceptually simple approach to the path planning problem is to generate a joint-space trajectory based on interpolation of a sequence of desired joint angles. In this approach, a number of knot points are chosen along the desired Cartesian path. The number of knots chosen is a tradeoff between exactness and computational expense. The Cartesian knots are then mapped into joint knots using inverse kinematics. Finally, an analytic interpolating curve is fit to the joint knots. This curve provides the path tracker with joint angles and derivatives at the controller rate.

The most popular type of interpolation is algebraic splines [2]. Higher order splines result in continuity of higher order derivatives, which reduces wear and tear on the robot, but this is at the expense of large oscillations of the trajectory. Polynomials with order as low as five (e.g. quartic splines) can overshoot extreme knots by as much as 60 degrees [16].

A recent development is the use of trigonometric polynomials to efficiently generate joint trajectories with little overshoot but continuous velocity, acceleration, and jerk. Trigonometric polynomials have the characteristic that if they are appropriately normalized in time, they are very smooth [11]. That is, the magnitude of the derivatives are relatively low, and the overshoot is relatively small. See [12] for a graphical contrast to algebraic splines. When piecewise continuous trigonometric polynomials are joined together, the result is called a *trigonometric spline*, hereafter referred to as a *TS*.

In this paper, it is shown that the use of TS's can result in a large savings of computational effort relative to algebraic splines. It is also shown that the use of TS's allows the path planning algorithm to proceed one segment at a time, so the path can be altered in mid-course if necessary (e.g. if an obstacle needs to be avoided).

Sometimes a specific path is desired for the robot. If TS's are interpolated between joint knots along the desired path, there

will in general be some error between the resultant Cartesian path and the desired Cartesian path. How does this error change as the number of knots changes? In this paper it is shown that the Cartesian path error is inversely proportional to the number of knots.

II. TRIGONOMETRIC SPLINES

The term *trigonometric spline* was first introduced by Schoenberg [10, 13]. But since then other definitions have appeared in the literature [6, 9]. So the term is not well-defined. In this section, the TS's used in this paper will be defined, and their application to robot path planning will be summarized. See [12] for details. The TS's used in this paper are those functions satisfying the following definition.

Definition 1 An m^{th} -order TS function $y(t)$ with a total of $2m$ constraints in each of the n closed arcs $[t_{i-1}, t_i]$ ($i = 1, \dots, n$) has the form

$$y(t) = y_i(t) \quad t \in [t_{i-1}, t_i] \quad (1)$$

where $y_i(t)$ is given by

$$y_i(t) = a_{i,0} + \sum_{k=1}^{m-1} (a_{i,k} \cos kt + b_{i,k} \sin kt) + a_{i,m} \sin m(t - \tau_i) \quad (2)$$

$$\tau_i = \sum_{j=0}^{2m-1} \tau_{i,j} / 2m \quad (3)$$

and $\tau_{i,j}$ are the values of t where $y_i(t)$ has a constraint applied.

The existence and uniqueness of these functions are asserted by the following theorem.

Theorem 1 If the TS functions of Definition 1 satisfy the property that, for each i and j , $y_i^{(r)}(\tau_{i,j})$ is not constrained unless $y_i^{(r-1)}(\tau_{i,j})$ is also constrained ($r = 1, 2, \dots$), then the TS functions exist and are unique.

Proof: The proof is long and complicated, and relies heavily on properties appearing in Schoenberg's original paper [10]. See Koch and Lyche [6, 7] for a proof. \square

A desired continuous time Cartesian trajectory can be discretized into $(n + 1)$ Cartesian goal points at times $t_0 < t_1 < \dots < t_n$. Inverse kinematics can be performed at each of these goal points, resulting in a set of $(n + 1)$ joint space goal points y_i for each joint. Then n fourth-order trigonometric polynomials $y_i(t)$ can be generated. Fourth-order polynomials are used so that the first three derivatives at each endpoint can be constrained. This allows the user to join the polynomials together

so as to have a joint-space path with continuous derivatives up to the third order. The function $y_i(t)$ ($i = 1, \dots, n$) is defined only on the time interval $[t_{i-1}, t_i]$. These n trigonometric polynomials are joined together to form a TS. Since $y_i(t)$ is a fourth-order trigonometric polynomial, it has eight undetermined coefficients (see Equation 2). The eight constraints used to determine the coefficients of $y_i(t)$ are

$$\begin{aligned} y_i(t_{i-1}) &= y_{i-1} \equiv y(t_{i-1}) \\ y_i(t_i) &= y_i \equiv y(t_i) \\ y_i^{(r)}(t_{i-1}) &= y_{i-1}^{(r)} \equiv y^{(r)}(t_{i-1}) \quad (r = 1, 2, 3) \\ y_i^{(r)}(t_i) &= y_i^{(r)} \equiv y^{(r)}(t_i) \quad (r = 1, 2, 3). \end{aligned} \quad (4)$$

The first two constraints of Equation 4 are given by the inverse kinematics solution of the Cartesian trajectory. There are several different ways to specify the last six constraints of Equation 4. One way is that the user may desire certain joint derivatives at the knots. Another possibility is that these constraints could be determined to minimize some objective function [15]. Yet another possibility is that these constraints could be chosen using some simple, heuristic method (see Section IIIA).

The determination of the eight coefficients for the spline segment $y_i(t)$ can be accomplished by the inversion of an 8×8 matrix (A_i) . But this matrix inversion can be performed a priori. It does not need to be performed in real time. The 8×8 matrix A_i is a function of only two parameters: t_{i-1} and t_i . So the time interval of each spline segment can be normalized to a fixed t_{i-1} and t_i . Then A_i is a known, constant matrix for all i , and A_i^{-1} is the same for each spline segment. Note that the invertibility of A_i is guaranteed by Theorem 1.

The normalized time interval of each spline segment must be small enough to prevent oscillations, but large enough to prevent numerical inaccuracies in A_i^{-1} . The exact time interval is not critical as long as it is between approximately 0.5 and 3.5. See [11] for a detailed analysis. In this paper, it will be assumed that $t_{i-1} = 0$ and $t_i = \pi/4$, ($i = 1, \dots, n$). Equation 4 then shows that for each spline segment we have four constraints at $t = 0$ and four constraints at $t = \pi/4$. Therefore the eight $\tau_{i,j}$'s in Equation 3 have the values $(0, 0, 0, 0, \pi/4, \pi/4, \pi/4, \pi/4)$. This results in $\gamma_i = \pi/8$ for all i . Equation 2 then becomes the familiar system

$$\begin{aligned} y_i(t) &= a_{i,0} + \sum_{k=1}^3 (a_{i,k} \cos kt + b_{i,k} \sin kt) + a_{i,4} \cos 4t \\ t &\in [0, \pi/4]. \end{aligned} \quad (5)$$

Equations 4-5 are used to determine the coefficients of the spline segments $y_i(t)$. The multiplication of the 8×8 constant matrix A_i^{-1} by an eight-element vector gives the eight coefficients of $y_i(t)$ as follows.

$$\begin{bmatrix} a_{i,0} & a_{i,1} & \dots & b_{i,3} & a_{i,4} \end{bmatrix}^T = \quad (6)$$

$$A_i^{-1} \begin{bmatrix} y_{i-1} & y_i & \dots & y_{i-1}''' & y_i''' \end{bmatrix}^T. \quad (7)$$

Note that the invertibility of A_i is guaranteed by Theorem 1. See Simon and Isik [12] for the numerical value of A_i . Since the segment $y_i(t)$ is defined on $t \in [0, \pi/4]$ for all i , the time-scaled TS $y(t)$ is given by

$$y(t + (i-1)\pi/4) = y_i(t), \quad t \in [0, \pi/4], \quad (i = 1, \dots, n). \quad (8)$$

The function $y(t)$ is a TS which satisfies the desired interpolation conditions, and which has length $n\pi/4$ seconds. The unscaled spline $\theta(t)$ given by

$$\theta(t) = y(n\pi t/4T) \quad t \in [0, T] \quad (9)$$

stretches the trajectory from its normalized length $n\pi/4$ to a desired length T .

III. COMPUTATIONAL EFFORT

Algebraic spline path planners are typically computationally inexpensive relative to optimal control algorithms. Nevertheless, the use of algebraic splines requires the specification of all knot angles and the solution of a set of linear equations before the robot motion can begin. This computational expense may or may not create a bottleneck at the path planning level, depending on the specific task and the available computational power.

It is claimed that the use of TS's results in a large savings of computational effort compared to the use of algebraic splines. This is because the use of TS's obviates the need for solving a linear system of equations. This section shows quantitatively that TS's are less computationally expensive than algebraic splines.

Computational expense can be characterized by the notion of flops (floating point operations) as used in Golub and Van Loan [5]. A multiply, division, addition, and subtraction are each one flop. The use of flops to quantify computational effort ignores the expense of assignment statements, bookkeeping associated with loops, and other costs. But flops can be used to get a rough idea of the computational expense of some algorithm.

In Section IIIA the computational effort required for the use of nominal TS's is derived. Nominal TS's are those TS's which are not optimized with respect to an objective function. Section IIIB outlines a real-time implementation of TS's. Section IIIC extends the real-time implementation to include the capability of obstacle avoidance. Section IIID derives the computational effort required for the use of quartic algebraic splines and presents a comparison between trigonometric and algebraic splines.

A. Nominal Trigonometric Splines

In this section, the computational expense of nominal TS's is derived. Nominal TS's are those which are not optimal in any sense. In other words, the free parameters of the TS (i.e. the interior knot derivatives) are chosen in a simple, heuristic manner.

Rearranging the Taylor series expansions of the time scaled TS $y(t)$ at t_{i-1} and t_{i+1} results in the following approximations for the first three derivatives of $y(t_i)$ (recall that the time between knots is $\pi/4$).

$$y_i' \approx (y_{i+1} - y_{i-1})/(\pi/2) \quad (10)$$

$$y_i'' \approx (y_{i+1} - 2y_i + y_{i-1})/(\pi/4)^2 \quad (11)$$

$$y_i''' \approx (y_{i+1}'' - y_{i-1}'')/(\pi/2) \quad (12)$$

These are the values of the normalized time derivatives which are used in Equation 6 in the formulation of nominal TS's. So the calculation of all $(n-1)$ normalized interior knot derivatives for one robot joint requires $4(n-1)$ adds and multiplies. It is seen from Equation 6 that the calculation of all n sets of spline segment coefficients for one robot joint requires $64n$ multiplies and $56n$ adds. Adding these numbers it is seen that the use of nominal TS's requires $[4(n-1) + 64n]$ multiplies and $[4(n-1) + 56n]$ adds, for a total of $(128n - 8)$ flops.

B. Real-time Implementation

As noted in Section II, the calculation of the TS segment coefficients are decoupled for each segment. That is, once a computer algorithm has the values of $y_0^{(r)}$ and $y_1^{(r)}$, the spline coefficients for the first segment can be calculated (see Equation 6). Then the robot can begin moving before any of the other spline segment coefficients are calculated. This is in contrast to most algebraic splines, where the coefficients for all the spline segments are calculated simultaneously through the solution of a

single set of linear equations. So TS's are quite amenable to real-time implementation. The spline coefficients for the first segment are calculated. These coefficients are used to calculate joint angles at the controller rate. Between each joint angle calculation, there is some idle time in the computer. During this idle time, a background routine is called to begin calculating the coefficients for the next spline segment. When another joint angle is needed by the controller during the first segment, the background routine is interrupted, and control is returned to the angle calculation routine. This process is depicted in Figure 1.

The computational savings of this implementation can be significant for large n (i.e. for many knots). Instead of performing $(128n - 8)$ flops before the robot begins to move, only 128 flops are required. The other flops are executed in background while the robot is already moving. So this implementation saves $(128n - 136)$ flops in initialization.

C. Obstacle Avoidance

While the robot is moving, some future knot angles may need to be changed in order to avoid an obstacle. The new knots can be input to the TS path planner in real-time. In practice, it would probably be the Cartesian knots which would be changed first. These Cartesian knots would be changed by an obstacle avoidance algorithm. Then the inverse kinematics routine would calculate the new corresponding joint-space knots. One restriction is that the new Cartesian knots must be calculated early enough so that the inverse kinematics routine has enough time to determine the new joint knots. That is, if knot m is changed by an obstacle avoidance routine, the inverse kinematics routine needs to calculate the new joint knot m before the robot reaches knot $(m - 1)$ in its movement. This real-time obstacle avoidance methodology is depicted in Figure 2, which is a generalization of Figure 1.

Of course, the generation of Cartesian knots subject to the constraint of obstacle avoidance is a complex issue in itself, and the focus of much current research [4]. In this section it is assumed that a collision detection and obstacle avoidance algorithm is in place, and that these modules provide the correct inputs to the TS trajectory planner.

Note that algebraic B-splines [16] also have a type of local control property. Changing one coefficient of a quartic B-spline results in the modification of three adjacent spline segments. Unfortunately, it cannot be known beforehand what new knot values will result. In order to change a knot to some desired value using B-splines, the entire linear system must be resolved.

So real-time obstacle avoidance is a feature which is not available in most algebraic spline trajectory planners. However, Lin and Chang [8] and Chand and Doty [2] have formulated algebraic splines which are based only on local knot information. They gain this advantage by sacrificing derivative continuity or by allowing joint angle errors at the knots. TS's are locally formulated without any such trade-off.

D. Comparison Between Trigonometric and Algebraic Splines

Algebraic quartic splines can be obtained by several different algorithms. Quartic splines are considered in this section because they provide the same number of continuous derivatives (three) as fourth-order TS's. The algorithm used by Thompson and Patel [16] is one of the more computationally efficient because it uses B-splines. The calculation of B-spline coefficients requires the solution of $(n + 10)$ simultaneous linear equations, where the coefficient matrix has a bandwidth of three. The bandlimited linear equation solver given in Gouib and Van Loan [5] requires $(n^2 + 33n + 190)$ multiplies and $(n^2 + 28n + 140)$ adds for a total of $(2n^2 + 61n + 330)$ flops.

The key issue in the comparison of trigonometric and algebraic splines is that the highest order term in the algebraic spline computational expense is n^2 , while the highest order term in the TS computational expense is n . That is,

$$\begin{aligned} \text{Algebraic Quartic Spline Expense} &= & (13) \\ (2n^2 + 61n + 330) &\approx O(n^2) \end{aligned}$$

$$\begin{aligned} \text{Nominal Trigonometric Spline Expense} &= & (14) \\ (128n - 8) &\approx O(n) \end{aligned}$$

where $(n + 1)$ is the number of knots. So for a large number of knots, a large savings of computational effort can be realized by using TS's. This is depicted in Figure 3. A close look at Figure 3 shows that if the number of knots is between eight and 28, algebraic splines are actually cheaper than TS's. However this savings is insignificant. The largest savings that could be realized by using algebraic splines rather than TS's is 223 flops (when $n = 17$). But when the number of knots reaches the order of 10^2 , the use of TS's results in a flops savings on the order of 10^4 . As the number of knots $(n + 1)$ increases, the savings increases with n^2 .

If the real-time implementation discussed in Section IIIB is used, an even larger savings of computational effort can be realized. This is because only 128 flops are required in the initialization phase of a TS path planner, while all $(2n^2 + 61n + 330)$ flops are required in the initialization phase of an algebraic quartic spline path planner. The comparison of initialization flops is depicted in Figure 4.

The number of knots actually encountered in practice is anywhere between five and 1000+. Tasks requiring a high degree of precision (e.g. laser welding or peg-in-the-hole operations), or tasks where a collision would be catastrophic (i.e. due to the high cost of the hardware involved) could necessitate the use of over 1000 knots. TS's would give the greatest computational advantage when used for such high-precision tasks. As robot technology is applied to more sensitive tasks in the future (e.g. medical work), the upper bound on the number of knots seen in practice is likely to increase. If the number of knots is 500, the use of quartic TS's results in an initial computational savings of over 500,000 flops.

IV. PATH ERROR ANALYSIS

In this section, the order of the error of TS interpolation will be derived. Sometimes a specific path is desired for the robot. If joint-based TS's are interpolated between several knots along the desired path, there will in general be some error between the resultant Cartesian path and the desired Cartesian path. In this section it is shown that the Cartesian path error is inversely proportional to the number of knots. The main result of this section is the following theorem.

Theorem 2 Let $\vec{f}^{(r)}(t)$ ($r = 0, 1, 2, 3$) be a desired fixed-time Cartesian path of a manipulator, along with its first three derivatives. Let $\vec{\theta}_i^{(r)}$ ($i = 0, \dots, n$) ($r = 0, 1, 2, 3$) be a sequence of joint angles and derivatives which agree kinematically with the Cartesian sequence $\vec{f}^{(r)}(t_i)$. Let $\vec{x}(t)$ be the Cartesian path which results when TS's are used to plan joint trajectories $\vec{\theta}(t)$ under the constraints $\vec{\theta}^{(r)}(t_i) = \vec{\theta}_i^{(r)}$. Then any norm of the Cartesian error between the desired path $\vec{f}(t)$ and the interpolated path $\vec{x}(t)$ is inversely proportional to n .

Proof: See Simon and Isik [11, 14]. □

The proof of the above theorem assumes that the constraints on the TS segments $y(t)$ exactly match the corresponding values of the desired joint trajectory segments $g(t)$. If a desired

path is given as a Cartesian function of time, inverse kinematics can be used to find the joint angles of that Cartesian path at specified knots. The Jacobian and its derivatives can be used to find the joint derivatives which correspond to the Cartesian path at specified knots.

V. SIMULATION RESULTS

In this section, the path error result which was derived analytically in Section IV is verified and extended numerically. The robot manipulator which is considered is a two-link robot with each link 0.5 meters long. The desired Cartesian space trajectory is assumed to be the following straight line.

$$\begin{aligned} x(t) &= (T-t)(\sqrt{2}/2)/T \text{ meters} & (15) \\ y(t) &= (\sqrt{2}/2)t/T \text{ meters} \end{aligned}$$

where T is the length in time of the desired trajectory. The initial and final configurations of the robot arm for this trajectory, and the corresponding desired joint trajectories, are shown in [14].

The number $(n+1)$ of evenly spaced Cartesian knots taken along the desired trajectory of Equation 18 was varied between four and 65. The interior knot derivatives for each TS corresponding to the desired Cartesian path were obtained using the Jacobian of the two-link robot given in Craig [3, page 171]. Since Theorem 2 in Section IV states that the RMS path error ϵ is inversely proportional to n , we can write

$$\begin{aligned} \epsilon &= k/n & (16) \\ \Rightarrow \log(\epsilon) &= \log(k) - \log(n) & (17) \end{aligned}$$

where k is an unknown constant. Therefore a log-log plot of ϵ versus n should show a straight line with a slope of minus one. This is exactly what we see in Figure 5.

It may not be practical for a user to compute the joint derivatives using the Jacobian and its inverse. With this in mind, the Cartesian path error was computed using heuristic joint derivatives (see Section IIIA). The results are given in Figure 6. It is seen that the path error is still inversely proportional to n . But a comparison with Figure 5 reveals that, as expected, the error for a given n is much worse than if the knot derivatives are chosen to agree with the desired Cartesian path.

VI. CONCLUSION

A general TS robot trajectory formulation algorithm has been summarized. The input to the algorithm is a sequence of $(n+1)$ angles for each joint which are determined by performing inverse kinematics on a sequence of Cartesian knots.

It has been shown that the computational expense of TS's is of the order n . This is in contrast to algebraic splines, whose computational expense is of the order n^2 . The computational savings resulting from the use of TS's is due to the fact that there is no need to solve a set of simultaneous linear equations.

In addition, it was shown that the decoupled nature of TS's makes them very amenable to real-time implementation and obstacle avoidance. An obstacle avoidance algorithm can operate in parallel with the TS path planner to prevent collisions while the robot is in motion. The locations of the obstacles do not need to be known beforehand, because they can be dealt with in real time. This is an important step toward making robots more autonomous, more robust, and less dependent on operator intervention.

This locally based spline formulation can also be accomplished (at least theoretically) using algebraic splines. But this results in eighth-order algebraic polynomials, which are twice the order of the required trigonometric polynomials. Algebraic polynomials of order greater than four have a well-known tendency to wander between knots [1]. Algebraic polynomials of

order eight are much too oscillatory for practical use. In fact, even fifth-order algebraic polynomials are too oscillatory for practical use [16]. The user of TS's, however, does not need to be concerned about oscillations between the knots. TS's can be formulated so as to be very smooth if the normalized time length of each spline segment is chosen appropriately.

It was shown both analytically and numerically that the error of the resulting Cartesian path approximation is inversely proportional to n . This result can be used to achieve any desired path accuracy without resorting to trial and error simulations.

REFERENCES

- [1] K. Atkinson, *An Introduction to Numerical Analysis*. New York, NY: John Wiley & Sons, 1989.
- [2] S. Chand and K. Doty, "On-Line Polynomial Trajectories for Robot Manipulators," *The International Journal of Robotics Research*, vol. 4, pp. 38-48, Summer 1985.
- [3] J. Craig, *Introduction to Robotics*. Reading, MA: Addison-Wesley, 1989.
- [4] L. Gwalia *et al.*, "Path Planning in the Presence of Vertical Obstacles," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 331-341, June 1990.
- [5] G. Golub and C. Van Loan, "Matrix Computations (Second Edition)," Baltimore, MD: The Johns Hopkins University Press, 1989.
- [6] P. Koch, "Error Bounds for Interpolation by Fourth Order Trigonometric Splines," in *Approximation Theory and Spline Functions*, ed. by Singh, Burry, and Watson, pp. 349-360, Dordrecht, Holland: D. Reidel Publishing Company, 1984.
- [7] P. Koch and T. Lyche, "Bounds for the Error in Trigonometric Hermite Interpolation," in *Quantitative Approximation*, ed. by Devore and Scherer, pp. 185-198, New York, NY: Academic Press, 1980.
- [8] C. Lin and P. Chang, "Joint Trajectories of Mechanical Manipulators for Cartesian Path Approximation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 1094-1102, Nov. 1983.
- [9] T. Lyche and R. Winther, "A Stable Recurrence Relation for Trigonometric B-Splines," *Journal of Approximation Theory*, vol. 25, pp. 266-279, March 1979.
- [10] I. Schoenberg, "On Trigonometric Spline Interpolation," *Journal of Mathematics and Mechanics*, vol. 13, pp. 795-825, 1964.
- [11] D. Simon, "A Unified Approach to Robot Path Planning Using Trigonometric Splines," Ph.D. Dissertation, Syracuse University, Syracuse, NY, Aug. 1991.
- [12] D. Simon and C. Isik, "Optimal Trigonometric Robot Joint Trajectories," *Robotica*, vol. 9, pp. 379-386, Oct. 1991.
- [13] D. Simon, "Optimal Robot Motions for Repetitive Tasks," *IEEE Conference on Decision and Control*, Tucson, AZ, pp. 3130-3134, Dec. 1992.
- [14] D. Simon and C. Isik, "Computational Complexity and Path Error Analyses of Trigonometric Joint Trajectories," *International Journal of Robotics and Automation*, vol. 7, no. 4, pp. 179-185, 1992.
- [15] D. Simon and C. Isik, "Suboptimal Robot Joint Interpolation Within User-Specified Knot Tolerances," *Journal of Robotic Systems*, vol. 10, no. 7, pp. 889-911, October 1993.
- [16] S. Thompson and R. Patel, "Formulation of Joint Trajectories for Industrial Robots using B-Splines," *IEEE Transactions on Industrial Electronics*, vol. IE-34, pp. 192-198, May 1987.

Figure 1 - Real-time implementation

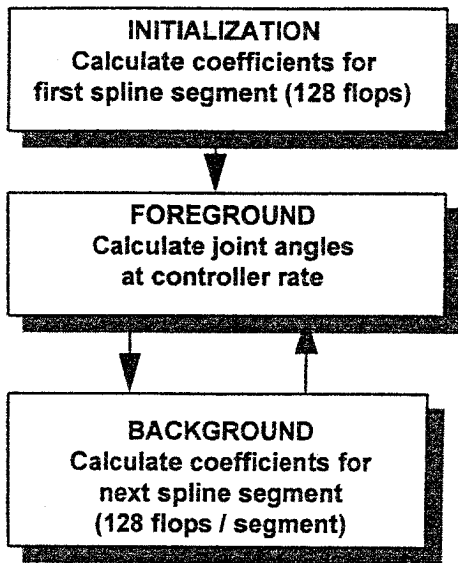


Figure 2 - Real-time obstacle avoidance implementation

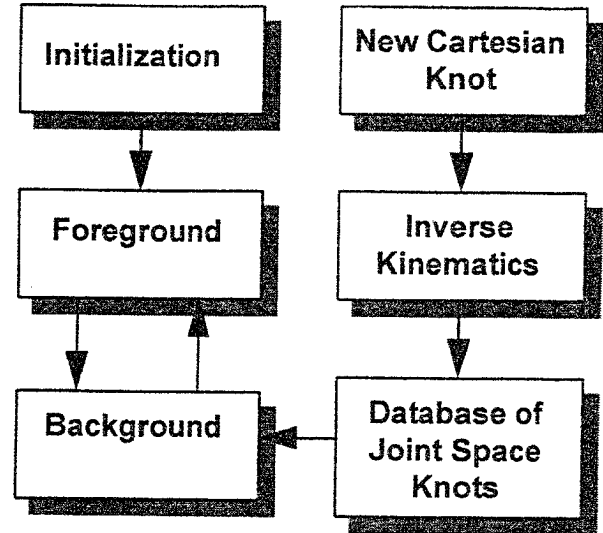


Figure 3 - Total computational effort of nominal splines

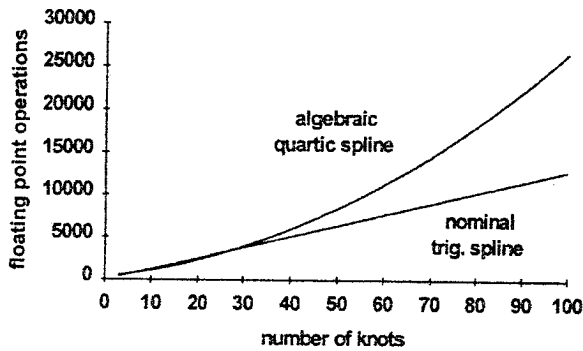


Figure 4 - Initial computational effort of nominal splines

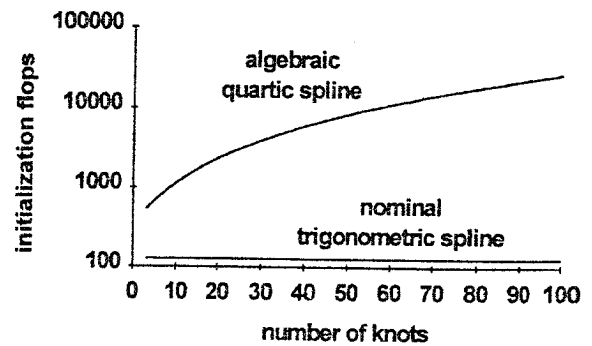


Figure 5 - Path error when knot derivatives agree with desired path

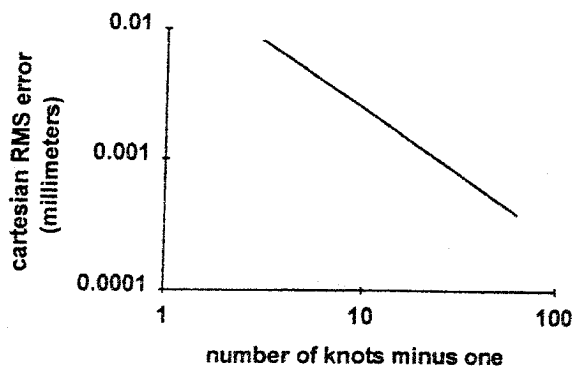


Figure 6 - Path error when heuristic knot derivatives are used

