

# COMPUTATIONAL COMPLEXITY AND PATH ERROR ANALYSES OF TRIGONOMETRIC JOINT TRAJECTORIES

D. Simon\* and C. Isik\*\*

## 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.

## Key Words

Robot path planning, trigonometric spline (TS), floating point operations (flops), path error

## 1. 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 trade-off 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 [1]. Higher-order splines result in continuity of higher-order derivatives, which reduces wear and tear on the robot, but 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° [2].

\* TRW, PO Box 1310, San Bernardino, CA 92402 USA

\*\* Department of Electrical Engineering, Syracuse University, Syracuse, NY 13244-1240 USA

(paper no. 91-020)

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 [3]. That is, the magnitude of the derivatives is relatively low and the overshoot is relatively small. See [4] 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 we show that the use of TSs can result in a large savings of computational effort relative to algebraic splines. It is also shown that the use of TSs 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 TSs 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? We show that the Cartesian path error is inversely proportional to the number of knots.

The paper is structured as follows. Section 2 gives a review of TSs and their application to robot path planning. Section 3 presents some results on computational expense and real-time obstacle avoidance and compares the expense of trigonometric and algebraic splines. Section 4 derives the Cartesian path error of joint-based TSs, and Section 5 provides a numerical verification and extension. Section 6 presents some concluding remarks.

## 2. Trigonometric Splines

The term *trigonometric spline* was first introduced by Schoenberg [5], but other definitions have since appeared in the literature [6, 7], so the term is not well defined. In this section, the TSs used in this paper will be defined and their application to robot path planning will be summarized (see [4] for details). The TSs 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

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

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

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 [5]. See Koch and Lyche [6, 8] for a proof. QED.

A desired continuous-time Cartesian trajectory can be discretized into  $(n+1)$  Cartesian goal points at time  $t_0 < t_1 < \dots < t_n$ . 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. Because  $y_i(t)$  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 (4) are given by the inverse kinematics solution of the Cartesian trajectory. There are several different ways to specify the last six constraints of (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 [4]. Yet another possibility is that these constraints could be chosen using some simple, heuristic method (see section 3.1).

The determination of the eight coefficients for the spline segment  $y_i(t)$  can be accomplished by the inversion of an  $8 \times 8$  matrix ( $A_i$ ), but this matrix inversion can be performed a priori—it does not need to be performed in real time. The  $8 \times 8$  matrix  $A_i$  is a function of only two parameters:  $t_{i-1}$  and  $t_i$ . Hence, 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 [3] for a detailed analysis). In this paper, we will assume 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 (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

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

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

$$\begin{bmatrix} a_{i,0} & a_{i,1} & \dots & b_{i,3} & a_{i,4} \end{bmatrix}^T = A_i^{-1} \begin{bmatrix} y_{i-1} & y_i & y_{i-1}' & \dots & y_{i-1}'' & y_i'' \end{bmatrix}^T. \quad (6)$$

Note that the invertibility of  $A_i$  is guaranteed by theorem 1 (see [4] for the numerical value of  $A_i$ ). Because 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

$$y(t + (i-1)\pi/4) = y_i(t), \quad t \in [0, \pi/4], \quad (i = 1, \dots, n). \quad (7)$$

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

$$\theta(t) = y(n\pi t/4T), \quad t \in [0, T] \quad (8)$$

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

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

Computational expense can be characterized by the notion of *flops* (floating point operations) as used in [9]. Multiplication, division, addition, and subtraction are each one flop. The use of flops to quantify computational effort ignores the expense of assignment statements, book-keeping 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 3.1 the computational effort required for the use of nominal TSs is derived. (Nominal TSs are those which are not optimized with respect to an objective function.) Section 3.2 outlines a real-time implementation of TSs; section 3.3 extends this real-time implementation to include the capability of obstacle avoidance; and section 3.4 derives the computational effort required for the use of quartic algebraic splines and presents a comparison between trigonometric and algebraic splines.

### 3.1 Nominal Trigonometric Splines

In this section we derive the computational expense of nominal TSs. 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  $\pi/4$ ):

$$y'_i \approx (y_{i+1} - y_{i-1}) / (\pi/2) \quad (9)$$

$$y''_i \approx (y_{i+1} - 2y_i + y_{i-1}) / (\pi/4)^2 \quad (10)$$

$$y'''_i \approx (y'_{i+1} - y'_{i-1}) / (\pi/2). \quad (11)$$

These are the values of the normalized time derivatives that are used in (6) in the formulation of nominal TSs. Hence, the calculation of all  $(n-1)$  normalized interior knot derivatives for one robot joint requires  $4(n-1)$  adds and multiplies. It can be seen from (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, we can see that the use of nominal TSs requires  $[4(n-1) + 64n]$  multiplies and  $[4(n-1) + 56n]$  adds, for a total of  $(128n-8)$  flops.

### 3.2 Real-Time Implementation

As noted in section 2, the calculation of the TS segment coefficients is 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 (6)). The robot can then 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, TSs 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.

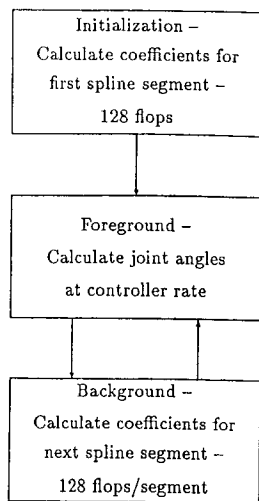


Figure 1. Real-time implementation.

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

### 3.3 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, the Cartesian knots would probably be changed first, by an obstacle avoidance algorithm, and the inverse kinematics routine would then 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 can 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 [10]. In this section we assume 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 [2] also have a type of local control property. Changing one coefficient of a quar-

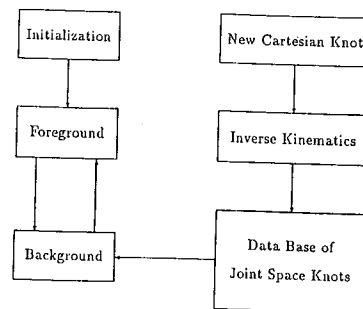


Figure 2. Real-time obstacle avoidance implementation.

tic 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.

For these reasons, real-time obstacle avoidance is a feature that is not available in most algebraic spline trajectory planners. However, Lin and Chang [11] and Chand and Doty [1] have formulated algebraic splines that are based only on local knot information. They gain this advantage by sacrificing derivative continuity or by allowing joint angle errors at the knots. TSs are locally formulated without any such trade-off.

### 3.4 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 TSs. The algorithm used by Thompson and Patel [2] 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 [9] 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  $n^2$ , and the highest-order term in the TS computational expense is  $n$ , that is,

$$\text{algebraic quartic spline expense} = (2n^2 + 61n + 330) \approx O(n^2) \quad (12)$$

$$\text{nominal trigonometric spline expense} = (128n - 8) \approx O(n) \quad (13)$$

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 TSs. This is depicted in Figure 3, a close

look at which shows that if the number of knots is between 8 and 28, algebraic splines are actually cheaper than TSs. However, this savings is insignificant. The largest savings that could be realized by using algebraic splines rather than TSs is 223 flops (when  $n=17$ ). But when the number of knots reaches the order of  $10^2$ , the use of TSs results in a flops savings on the order of  $10^4$ . As the number of knots  $(n+1)$  increases, the savings increases with  $n^2$ .

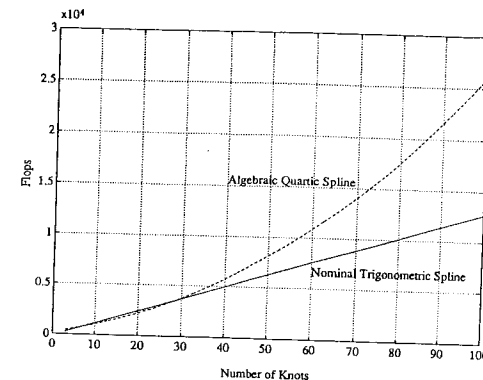


Figure 3. Total computational effort of nominal splines.

If the real-time implementation discussed in Section 3.2 is used, an even larger savings of computational effort can be realized, because only 128 flops are required in the initialization phase of a TS path planner, whereas 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.

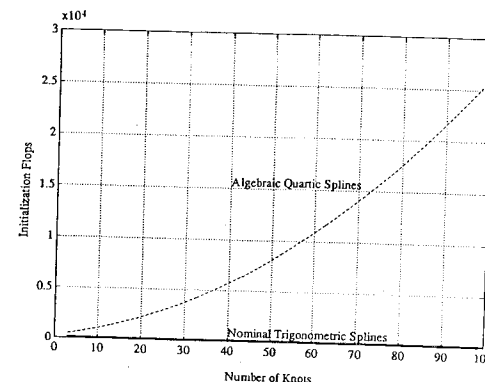


Figure 4. Initial computational effort of nominal splines.

The number of knots actually encountered in practice is anywhere between 5 and 1,000+. 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 (owing to the high cost of the hardware involved) could necessitate the use of over 1,000 knots. TSs 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 TSs results in an initial computational savings of over 500,000 flops.

#### 4. 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 TSs 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 we show that the Cartesian path error is inversely proportional to the number of knots. We begin with the following lemma.

**Lemma 1.** Given a desired normalized time function  $g(t)$ ,  $t \in [0, h]$  and the TS segment  $y(t)$ , which has  $2m$  constraints (see definition 1 in section 2), the error between the two can be given by

$$g(t) - y(t) = 2c(h, t) L_{2m} g(\xi) \prod_{j=1}^{2m} \sin\left(\frac{t - \tau_j}{2}\right) / (2m)! \quad (14)$$

where  $h$  is the normalized time length,  $t \in [0, h]$ ,  $\xi$  is some constant in the interval  $[0, h]$ ,  $\tau_j$  are the  $2m$  values of  $t$  where  $y(t)$  has a constraint applied, and  $c(h, t)$  is a function that is independent of  $g(t)$ . The differential operator  $L_{2m} g(\xi)$  is given by

$$L_{2m} g(\xi) = \left[ \cos\left(\frac{t - \xi}{2}\right) D + m \sin\left(\frac{t - \xi}{2}\right) \right] L_{2m-1} g(\xi) \quad (15)$$

and  $L_{2m+1}$  is given by

$$L_{2m+1} g(\xi) = D(D^2 + 1^2) \dots (D^2 + m^2) g(\xi), \quad D \equiv d/d\xi \quad (16)$$

*Proof:* The proof is presented in [8]. QED.

We are now in a position to state and prove the main result of this section.

**Theorem 2.** Let  $\vec{y}^{(r)}(t)$  ( $r = 0, 1, 2, 3$ ) 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 that agree kinematically with the Cartesian sequence  $\vec{y}^{(r)}(t)$ . Let  $\vec{x}(t)$  be the Cartesian path that results when TSs 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  $\vec{y}(t)$  and the interpolated path  $\vec{x}(t)$  is inversely proportional to  $n$ .

*Proof:* We begin by applying Lemma 1 to the robot path planning problem.  $g(t)$  is the desired joint trajec-

tory between two given knots, and  $y(t)$  is the TS segment that approximates  $g(t)$ . Both  $g(t)$  and  $y(t)$  are normalized so that  $t \in [0, \pi/4]$ . Therefore  $y^{(r)}(0) = g^{(r)}(0)$  and  $y^{(r)}(\pi/4) = g^{(r)}(\pi/4)$ , ( $r = 0, 1, 2, 3$ ). In the application of lemma 1 to robot path planning,  $m = 4$ ,  $h = \pi/4$ , and the eight values of  $\tau_j$  are  $(0, 0, 0, 0, \pi/4, \pi/4, \pi/4, \pi/4)$ . We thus obtain

$$g(t) - y(t) = c_1(t) \left[ \cos\left(\frac{t - \xi}{2}\right) D + 4 \sin\left(\frac{t - \xi}{2}\right) \right] D \prod_{k=1}^3 (D^2 + k^2) g(\xi) \quad (17)$$

for some constant  $\xi$  in the interval  $[0, \pi/4]$ . But note that the desired real-time joint trajectory segment  $\theta(t)$  is given by

$$\theta(4Lt/\pi) = g(t) \quad t \in [0, \pi/4] \quad (18)$$

where  $L$  is the length in real time between the two knots under consideration (compare with (8)). It can therefore be seen that

$$D^r g(\xi) = (L/h)^r D^r \theta(L\xi/h) \quad (19)$$

where  $h$  is the constant  $\pi/4$ , and we obtain

$$g(t) - y(t) = c_1(t) \left\{ \cos\left(\frac{t - \xi}{2}\right) [(L/h)^8 D^8 \theta(\gamma) + 14(L/h)^6 D^6 \theta(\gamma) + 49(L/h)^4 D^4 \theta(\gamma) + 36(L/h)^2 D^2 \theta(\gamma)] + 4 \sin\left(\frac{t - \xi}{2}\right) \times [(L/h)^7 D^7 \theta(\gamma) + 14(L/h)^5 D^5 \theta(\gamma) + 49(L/h)^3 D^3 \theta(\gamma) + 36(L/h) D \theta(\gamma)] \right\} \quad (20)$$

where again  $\xi$  is some constant in the interval  $[0, \pi/4]$  and  $\gamma = 4\xi L/\pi$ . This equation shows that the joint path error of the spline segment under consideration is of the order  $L$ :

$$g(t) - y(t) \approx \mathcal{O}(L). \quad (21)$$

Equation (21) is valid for each of the  $n$  joint trajectory segments. It is therefore valid for the entire trajectory as a whole. This implies that

$$\|\vec{g}(t) - \vec{y}(t)\| \approx \mathcal{O}(L) \quad (22)$$

where  $\vec{g}(t)$  and  $\vec{y}(t)$  are the vectors of desired and approximate joint paths, and  $\|\cdot\|$  is any vector norm. Lin and Chang [11] show that if each joint angle error is of the order of  $L$ , then the resulting Cartesian path error is also of the order of  $L$ . Assuming that the desired path is fixed-time, the time between knots is inversely proportional to  $n$ . We thus obtain

$$\|\vec{g}(t) - \vec{y}(t)\| \approx \mathcal{O}(1/n). \quad (23)$$

This concludes the proof of theorem 2. QED.

The above analysis 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 that correspond to the Cartesian path at specified knots.

#### 5. Simulation Results

In this section, the path error result that was derived analytically in section 4 is verified and extended numerically. The robot manipulator that is considered is a two-link robot with each link being 0.5 meters long. The desired Cartesian space trajectory is assumed to be the following straight line:

$$\begin{aligned} x(t) &= (T - t)(\sqrt{2}/2)/T \text{ meters} \\ y(t) &= (\sqrt{2}/2)t/T \text{ meters} \end{aligned} \quad (24)$$

where  $T$  is the length in time of the desired trajectory. The initial and final configurations of the robot arm for this trajectory are plotted in Figure 5. The corresponding desired joint trajectories  $\theta_1(t)$  and  $\theta_2(t)$  are depicted in Figure 6.

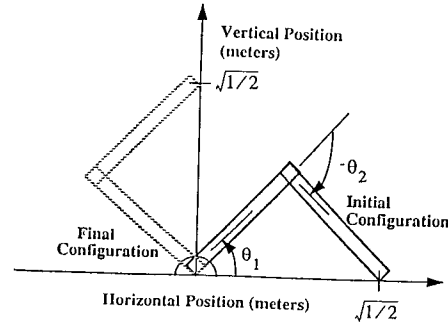


Figure 5. Initial and final configurations of straight-line Cartesian path.

The number  $(n + 1)$  of evenly spaced Cartesian knots taken along the desired trajectory of (24) was varied between 4 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 [12, p. 171]. Because theorem 2 in section 4 states that the RMS path error  $\epsilon$  is inversely proportional to  $n$ , we can write

$$\epsilon = k/n \quad (25)$$

$$\Rightarrow \log(\epsilon) = \log(k) - \log(n) \quad (26)$$

where  $k$  is an unknown constant. Therefore, a log-log plot

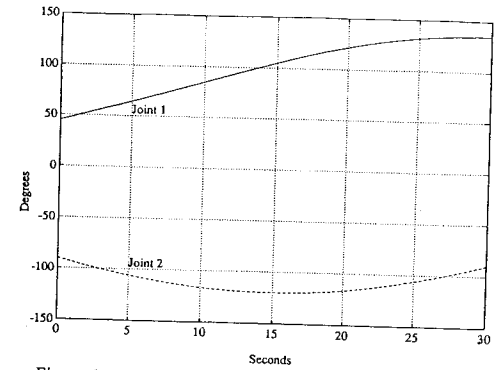


Figure 6. Joint trajectories corresponding to straight-line Cartesian path.

of  $\epsilon$  versus  $n$  should show a straight line with a slope of  $-1$ . This is exactly what we see in Figure 7.

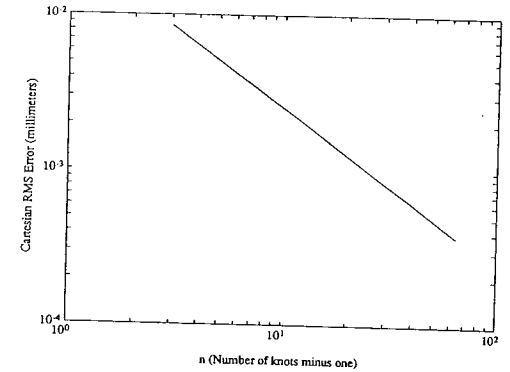


Figure 7. Path error when knot derivatives agree with desired path.

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 3.1). The results are given in Figure 8. It is seen that the path error is still inversely proportional to  $n$ , but a comparison with Figure 7 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.

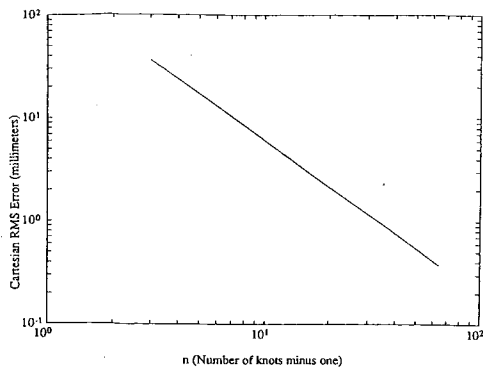


Figure 8. Path error when heuristic knot derivatives are used.

## 6. 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 TSs 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 TSs 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 TSs 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 [13]. 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 [2]. The user of TSs, however, does not need to be concerned

about oscillations between the knots; TSs 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.

## Acknowledgments

The authors thank Dr. Neville Maxwell of JPL for sharing his knowledge of robot path planning applications, and the anonymous reviewers for their suggestions for improvements.

## References

- [1] S. Chand & K. Doty, "On-Line Polynomial Trajectories for Robot Manipulators," *Int. J. of Robotics Research*, 4, 38-48.
- [2] S. Thompson & R. Patel, "Formulation of Joint Trajectories for Industrial Robots using B-Splines," *IEEE Trans. on Industrial Electronics*, IE-34, 192-199.
- [3] D. Simon, *A Unified Approach to Robot Path Planning Using Trigonometric Splines*, doctoral diss., Syracuse University, Syracuse, NY, USA, 1991.
- [4] D. Simon & C. Isik, "Optimal Trigonometric Robot Joint Trajectories," *Robotics*, 9, 379-386.
- [5] I. Schoenberg, "On Trigonometric Spline Interpolation," *J. of Mathematics and Mechanics*, 13, 795-825.
- [6] P. Koch, "Error Bounds for Interpolation by Fourth Order Trigonometric Splines," in S. Singh, J. Burry, & B. Watson, ed., *Approximation Theory and Spline Functions* (Dordrecht: D. Reidel, 1984), 349-360.
- [7] T. Lyche & R. Winther, "A Stable Recurrence Relation for Trigonometric B-Splines," *J. of Approximation Theory*, 25, 266-279.
- [8] P. Koch & T. Lyche, "Bounds for the Error in Trigonometric Hermite Interpolation," in R. DeVore & K. Scherer, ed., *Quantitative Approximation* (New York: Academic Press, 1980), 185-196.
- [9] G. Golub & C. Van Loan, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press, 1989).
- [10] L. Gewali, S. Ntafon, & I. Tollis, "Path Planning in the Presence of Vertical Obstacles," *IEEE Trans. on Robotics and Automation*, 6, 331-341.
- [11] C. Lin & P. Chang, "Joint Trajectories of Mechanical Manipulators for Cartesian Path Approximation," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13, 1094-1102.
- [12] J. Craig, *Introduction to Robotics* (Reading, MA: Addison-Wesley, 1989).
- [13] K. Atkinson, *An Introduction to Numerical Analysis* (New York: Wiley, 1989).



member of Eta Kappa Nu and a senior member of the I.E.E.E.

Can Isik received his Ph.D. in Electrical Engineering from the University of Florida in 1985. He then joined the Electrical and Computer Engineering Department at Syracuse University where he is currently an associate professor. His academic interests include robotics, controls, and applications of fuzzy logic and neural networks. He is a



search interest include robotics, manufacturing systems, and laser metrology.

Lu-Sin Liu received the B.E. degree in Industrial Education from the National Chang-Hua Normal University in 1985 and the M.S. degree in Mechanical Engineering from the National Taiwan Institute of Technology in 1991. He is currently a lecturer in the Department of Mechanical Engineering at China Junior College of Technology. His re-



search interest include robotics, manufacturing systems, and laser metrology.

René V. Mayorga received his Ph.D. in Electrical Engineering in 1981 from the University of Waterloo. He is currently a Research Assistant Professor in the Department of Systems Design Engineering at the University of Waterloo. For the past several years he has been in the PAMI group involved in some research projects for computer integrated/flexible manufacturing systems. Lately, he has been participating in three STEAR (Space Station, Strategic Technologies in Automation and Robotics) for the Canadian Space Agency. He has authored and co-authored chapters/sections in several engineering books/monographs and published numerous articles in scientific journals and conference proceedings. His current research interests are on: robot manipulator design analysis/optimization and trajectory/path planning; robot kinematic redundancy and singularities avoidance; and coordination and control of multi-robot (autonomous and teleoperated) systems.



University of Waterloo, Canada. He joined the Department of Robotics and Automation at the Tecnopolis - CSATA Novus Ortus Scientific Park, Bari, Italy in 1990. Since then, he has been involved in several space projects for the Italian Space Agency and some aerospace companies. His research interests include Man-Machine Interface for telerobotic systems, advanced robotics in partially structured environments, and 3D simulation for space telerobotics.

Nico Milano graduated in Computer Science in 1986 from the University of Bari, Italy. He received a fellowship in 1986 from the Scientific Research Center of IBM, Rome, to investigate AI techniques for manipulator task planning. In the fall of 1989 he was a visiting scholar in the PAMI Group of the Department of Systems Design Engineering, at the

Huei Peng received his B.S. degree in Mechanical Engineering from the National Taiwan University, Taipei, in 1984. He received his M.S. and Ph.D. degrees in Mechanical Engineering from the Pennsylvania State University and the University of California at Berkeley in 1988 and 1992, respectively. He is currently an assistant research engineer in the PATH program of UC Berkeley. His research interests include robust control, adaptive and learning control for nonlinear systems, with emphasis on their real-time applications to vehicular and robotic systems.



Jahangir Rastegar received his B.S. degree from the Southern Methodist University in 1970, and M.S. and Ph.D. degrees from the Stanford University in 1972 and 1977 respectively. He is currently Associate Professor of Mechanical Engineering at the State University of New York at Stony Brook. His research interests are in the areas of kinematics, dynamics and control of mechanical systems.