1997-05-01

On-line Cartesian trajectory control of mechanisms along complex curves

Edward Red

Zhaoxue Yang

Original Publication Citation

Robotica 15.3 (May 1997): 263-274.

Robotica (1997) volume 15, pp 263–274. Printed in the United Kingdom © 1997 Cambridge University Press

On-line Cartesian trajectory control of mechanisms along complex curves

*Zhaoxue Yang and †Edward Red

(Received in Final Form: July 25, 1996)

SUMMARY

New methods have been developed to control a mechanism's realtime Cartesian motion along spatially complex curves such as Non-Uniform Rational B-splines (NURBS). The methods dynamically map the critical trajectory parameters between parameter space, Cartesian space, and joint space. Trajectory models that relate Cartesian tool speeds and accelerations to joint speeds and accelerations have been generalized so that they can be applied to most classes of robots and CNC mechanisms.

A simple and efficient predictor-corrector method uses finite difference theory to predict the parametric changes required to generate the desired curvilinear distances along the trajectory, and then correct the erorrs arising from this prediction. Polynomial approximation methods successfully approximate joint speeds and accelerations rather than require a closed-form inverse Jacobian solution.

The numerical algorithms prove to be time bounded (fixed number of computational steps), and the generated trajectories are smooth and continuous. Both simulation and physical experiments using an Open-Architecture Controller demonstrate the feasibility and usefulness of the developed trajectory generation algorithms and methods. The methods can be conducted at trajectory rates greater than 100 Hz, depending on mechanism complexity.

KEYWORDS: Parametric paths; Cartesian trajectory generation; Complex curves.

INTRODUCTION

Cartesian trajectory generation using lines and arcs has long been considered by researchers such as Paul[1] and Taylor.[2] Previously, researchers such as Lin,[3] Thompson,[4] Chang,[5] and Aken[6] used spline functions to construct joint trajectories which approximated complex curves specified in Cartesian space. These methods were typically applied off-line. Practical on-line Cartesian trajectory generation for complex curves such as NURBS (Non-Uniform Rational B-Spline) posed a more difficult problem.

Froissart[7] was one of the first researchers to apply a realtime complex trajectory algorithm in Cartesian space. The method decomposed the geometric trajectory into two paths, one in position, one in orientation, and then distributed the orientation motion in such a way that the resulting orientation synchronized with the position. A Bezier representation computed a fifth degree polynomial to fit the prescribed path which guaranteed continuous velocity and acceleration. But the method developed was limiting to planar curves, and used a Newton-Raphson method requiring unpredictable time for convergence.

In practical applications such as gluing, painting, and trimming, the initial and terminal poses, and sometimes several intermediate poses, are commonly specified. These poses are usually connected by straight lines, or possibly by a combination of straight lines and circular arcs which we call a simple curve. Because the segments of the simple curve can be expressed as explicit functions, it is easy to calculate the length changes along the curve segment, and the time derivatives. For example, a circular arc can be expressed as $f_x(u) = R\cos u$ and $f_y(u) = R\sin u$, where R is the radius of the arc, and u is the parameter which represents the arc angle. The first derivatives of the circular arc with respect to u are $df_x(u)/du = -R\sin u$ and $df_y(u)/du = R\cos u$, and the length of the arc segment is $R\,u$.

A curve which is generated by a parametric spline function, such as a NURBS or Bezier curve, is referred to as a complex curve. Common practice decomposes, off-line, the complex curve into simpler shapes such as lines and circular arcs – see Beazel's[8] procedure. It is difficult to establish the Cartesian trajectory relationships from the curve parameters, such as the length changes along the curve, or the derivatives and their relationships to time, because the length of a B-spline segment involves numerical integration. The derivatives are also complex as compared with simple curves.

Off-line planning is adequate as long as the task requirements remain constant, because a trajectory, once computed, is generally difficult to modify in response to changes or realtime sensor information. For example, in NC machining tool paths are usually pre-processed, because NC paths, once defined, are generally fixed. But pre-processed paths limit the integration of sensor information into machine tool control. Yet, unpredictable changes in the required task, such as tool wear, occur commonly in industrial NC applications.

With the increase in computing speeds and the open architectures provided by systems like the Robline System, it is now feasible to provide a dynamic trajectory

* CIMETRIX, Inc.
†Dept. of Manufacturing Engineering and Engineering Technology Brigham Young University, Provo, Utah 84602 (USA).

generator that can process motion along complex curves. This paper demonstrates how such a trajectory generator can be organized, by

- developing a model that relates the motion parameters in Cartesian, parameter, and joint space for complex curves.
- developing bounded numerical algorithms to generate Cartesian moves along a NURBS path, using a fixed number of algorithmic calculations, and thus bounding the calculations in time.
- transforming the parameters created in Cartesian space into joint space where manipulator control is performed.
- integrating the software into an open-architecture simulation and control system called Robline.
- demonstrating realtime physical control of the trajectory along both 2-D and 3-D Cartesian NURBS curves using an actual robot.

TRAJECTORY METHODS

Figure 1 introduces the six primary trajectory generation steps implemented for simulation and actual control of mechanisms:

1. **initialization** of curve length, tool orientation, and other parameters.
2. Cartesian **trajectory prediction** for step length and trajectory speed and acceleration.
3. **parameter prediction.**
4. **correction** of step length, curve parameter, and step time.

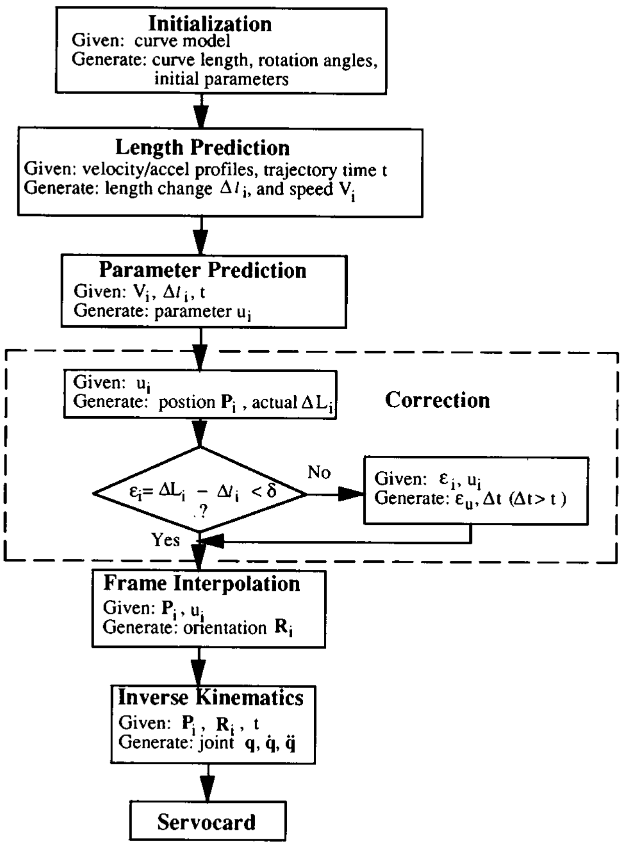
Initialization
Given: curve model
Generate: curve length, rotation angles, initial parameters
Length Prediction
Given: velocity/accel profiles, trajectory time t
Generate: length change Δl_i, and speed V_i
Parameter Prediction
Given: V_i, Δl_i, t
Generate: parameter u_i
Given: u_i
Generate: postion P_i, actual ΔL_i
Correction
$\varepsilon_i = \Delta L_i - \Delta l_i < \delta$?
No
Given: ε_i, u_i
Generate: ε_u, Δt ($\Delta t > t$)
Yes
Frame Interpolation
Given: P_i, u_i
Generate: orientation R_i
Inverse Kinematics
Given: P_i, R_i, t
Generate: joint q, $\dot{q}$, $\ddot{q}$
Servocard

Fig. 1. Trajectory generation flowchart.

5. Cartesian frame **interpolation.**
6. joint space **inverse kinematics** and joint speed estimates.

1 Initialization

Length L along a complex curve, specified by parameter u, is determined by the integral

$$L = \int_{u_1}^{u_2} \left| \frac{d\mathbf{P}(u)}{du} \right| u \tag{1}$$

where $\mathbf{P}(u) = (x(u), y(u), z(u))$ is a complex curve such as a NURBS or Bezier curve (see Appendix). The reader is referred to Choi,[9] Farin,[10] and Mortenson[11] for introductions to the basic theory of differential and B-spline geometry. A brief review is also included in the Appendix to this paper.

Gaussian quadrature[12] is concerned with optimizing integral evaluation by reducing the function evaluations to yield high accuracy. The procedure selects values $x_1, x_2, \ldots, x_n$ in the interval $[a, b]$, and constants $w_1, w_2, \ldots, w_n$, to minimize the error obtained in performing the approximation

$$\int_a^b \mathrm{f(x)\,dx} = \sum_{i=0}^{n} w_i f(x_i) + \varepsilon(f, n) \tag{2}$$

for an arbitrary function $f(x)$, where $\varepsilon(f, n)$ represents the error term,

$$\varepsilon(f, n) = \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b [\varphi_n(x)]^2 w(x)\, dx = c_n f^{(2n)}(\xi) \tag{3}$$

and where $\xi \in (a, b)$, $\varphi_n(x) = \prod_{i=1}^{n} (x - x_i)$. Constant c_n can be determined by applying (3) to polynomials of degree $2n$.

We first restrict consideration to the normalized interval $[-1, 1]$. For any $n \geq 1$, our objective is to find n sample points $x_1, x_2, \ldots, x_n$ in $[-1, 1]$ and n corresponding weights $w_1, w_2, \ldots, w_n$ such that the n-point Gaussian quadrature formula on $[-1, 1]$, namely

$$\int_{-1}^{1} f(x)\, dx \approx w_1 f(x_1) + w_2 f(x_2) + \cdots + w_n f(x_n) \tag{4}$$

approximates the integration of $f(x)$ which is exact for polynomials of degree $\leq 2n - 1$. We call x_i the Gaussian sample points of $[-1, 1]$, and w_i their Gaussian weights. If we make (4) exact for $f(x) = 1, x, x^2, \ldots, x^{2n-1}$, we generate $2n$ equations:

$$\int_{-1}^{1} x^{i-1}\, dx = w_1 x_1^{i-1} + w_2 x_2^{i-1} + \cdots + w_n x_n^{i-1} \qquad (i = 1, 2, \ldots, 2n) \tag{5}$$

The system of equations in (5) is nonlinear in the $2n$ variables $x_1, x_2, \ldots, x_n$ and $w_1, w_2, \ldots, w_n$. Its solution generally requires a numerical procedure, but these

Table I. Gaussian quadrature abscissas and weights.

n	Abscissas $x_{n,i}$	Weights $w_{n,i}$	Errors $\varepsilon(f, n)$
2	−0.5773502692	1.0000000000	$\frac{f^{(4)}(\xi)}{135}$
	0.5773502692	1.0000000000	
3	±0.7745966692	0.5555555556	$\frac{f^{(6)}(\xi)}{15750}$
	0.000000000	0.8888888889	
4	±0.8611363116	0.3478548451	$\frac{f^{(8)}(\xi)}{3472875}$
	±0.339810436	0.6521451549	
5	±0.9061798459	0.2369268851	$\frac{f^{(10)}(\xi)}{1237732650}$
	±0.5384693101	0.4786286705	
	0.0000000000	0.5688888889	
6	±0.9324695142	0.1713244924	$\frac{f^{(12)}(\xi)2^{13}(6!)^4}{(12!)^3 13!}$
	±0.6612093865	0.360715730	
	±0.2386191861	0.4679139346	

constants have been tabulated and are easily available. Table I lists the values up to six points and the error term $\varepsilon(f, n)$ which can be used to determine the accuracy of the Gaussian quadrature integration.[13]

To integrate $f(u)$ over the arbitrary interval $[a, b]$ using Gaussian quadrature, map the x interval $[-1, 1]$ into the u interval $[a, b]$ using the linear transformation

$$u = a + (x + 1)(b - a)/2; \quad du = \frac{(b-a)}{2}\,dx \tag{6}$$

to obtain

$$\int_a^b f(u)\,du = \frac{b-a}{2}\int_{-1}^{1} f\left(a + \frac{b-a}{2}(x+1)\right)dx \tag{7}$$

If we use a Gaussian formula on $[-1, 1]$ to approximate integral (7), we get

$$\int_a^b f(u)\,du \approx \frac{b-a}{2}\left[w_1 f(u_1) + w_2 f(u_2) + \cdots + w_n f(u_n)\right] \tag{8}$$

where w_i is the tabulated Gaussian weight associated with the tabulated Gaussian sample point x_i in $[-1, 1]$, and u_i is obtained from x_i as follows:

$$u_i = a + (x_i + 1)(b - a)/2, \quad (i = 1, \ldots, n) \tag{9}$$

The general n-point Gaussian quadrature rule is exact for polynomials of degree $\leq 2n - 1$. To integrate over large intervals, we must apply the large point Gaussian quadrature formula to achieve the desired accuracy according to the error formula (3). An alternative method first divides the large interval into small sub-intervals according to the interval and error formula, and then uses the Gaussian quadrature technique to integrate each sub-interval. During this process, a table of subdivision points, (u_i, l_i) where l_i is the arc length from parameter u_{i-1} to u_i, is created. After the table is built, all subsequent arc length calculations are greatly accelerated by using the table to find the region in which arc length is to be calculated.

When calculating the length from parameter u_0 to u, the table is searched to find the region $[u_i, u_{i+1}]$ such that $u_i < u < u_{i+1}$. The arc length from u_i to u is then calculated quickly by using the Gaussian quadrature technique.

2 Trajectory prediction

A simple trapezoidal velocity profile uses constant accelerations and decelerations to change the desired speed. For short moves the attained (peak) speed is less than the desired speed and the velocity profile assumes a triangular shape. But for most moves, given initial speed, final speed, desired speed, acceleration/deceleration, and trajectory step time, the trajectory distance over a discrete time step can be predicted by considering three motion stages: rise motion, steady motion and fall motion. The special cases that complicate the implementation of these equations are considered by Yang.[14]

Another commonly used trajectory generator uses constant jerk to control the velocity profile. Jerk, the time derivative of acceleration, can be specified such that it creates the desired acceleration and velocity at each point of the trajectory. Yang[14] describes the implementation of this profile.

3 Parameter prediction

Because the tool pose motion is dependent on the parameter u, and the velocity is related to the arc-length L, we must relate u to L for planning the pose and velocity motion profiles across complex curves. Since numerical procedures like Newton-Raphson are somewhat unpredictable in their convergence, they can only be used for off-line trajectory generation. The predictor-corrector method introduced in this section requires a bounded time for the necessary calculations.

For a complex curve, we first create a table of parameter/length pairs (u_i, l_i) using the trajectory profiles discussed earlier, where l_i is the arc-length corresponding to curve parameter u_i. We then build a piecewise polynomial interpolation function $u = f(l)$, where f is the polynomial function. From this function, we can predict the initial changes of the parameter (du) corresponding to changes in the arc-length (dl) using a cubic spline interpolant.[12]

After we predict the first three step changes of the parameter u for predicted arc-length changes, we can use quadratic or cubic extrapolation methods to predict the next parameter change for a given length change. For each step prediction, we update the prediction model by using the current value and the past two or three step values.

Suppose that the function $u = f(l)$ is known at the three points (l_0, u_0), (l_1, u_1), (l_2, u_2), where the values l_i satisfy $l_0 < l_1 < l_2$ and $u_i = f(l_i)$. A quadratic polynomial $P(l)$ of degree 2 can be constructed which passes through these 3 points. When $l_0 < l < l_2$ the approximation $P(l)$ is called an interpolated value. If either $l < l_0$ or $l_2 < l$, then $P(l)$ is an extrapolated value. The quadratic curve $u = P(l)$ that passes through the three points (l_0, u_0), (l_1, u_1), and (l_2, u_2) where l_0, l_1, l_2 are distinct, has the form

$$P(l) = u_0\frac{(l-l_1)(l-l_2)}{(l_0-l_1)(l_0-l_2)} + u_0\frac{(l-l_0)(l-l_2)}{(l_1-l_0)(l_1-l_2)} + u_2\frac{(l-l_0)(l-l_1)}{(l_2-l_0)(l_2-l_1)} \tag{10}$$

We assume that $f(l)$ is continuous on an interval $[a, b]$ containing the distinct values $l_i (i = 0, 1, 2)$. Thus $f(l) = P(l) + E(l)$, where $E(l)$ represents the approximation error term. Because the l_i represent the moving length, they are distinct automatically. If the derivatives up to order 3 are continuous, then there exists a value $\xi \in (a, b)$ such that

$$E(l) = (l-l_0)(l-l_1)(l-l_2)\frac{f^{(3)}(\xi)}{3!} \tag{11}$$

The cubic curve $u = P(l)$ that passes through the four points (l_0, u_0), (l_1, u_1), (l_2, u_2) and (l_3, u_3) where l_0, l_1, l_2, l_3 are distinct, has the form

$$\begin{aligned} P(l) = {} & u_0\frac{(l-l_1)(l-l_2)(l-l_3)}{(l_0-l_1)(l_0-l_2)(l_0-l_3)} \\ & + u_1\frac{(l-l_0)(l-l_2)(l-l_3)}{(l_1-l_0)(l_1-l_2)(l_1-l_3)} \\ & + u_2\frac{(l-l_0)(l-l_1)l-l_3)}{(l_2-l_0)(l_2-l_1)(l_2-l_3)} \\ & + u_3\frac{(l-l_0)(l-l_1)(l-l_2)}{(l_3-l_0)(l_3-l_1)(l_3-l_2)} \end{aligned} \tag{12}$$

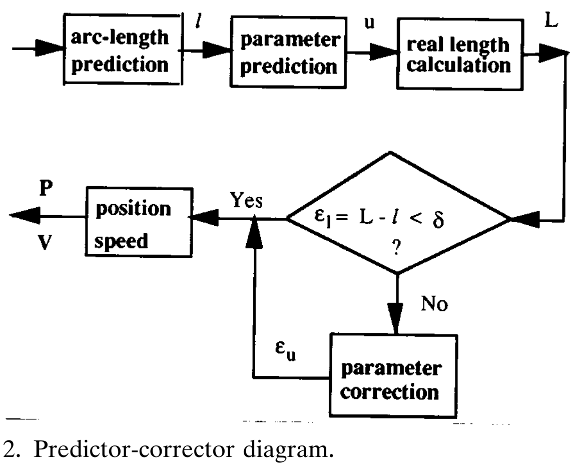

Fig. 2. Predictor-corrector diagram.

The error term in the cubic approximation is

$$E(l) = (l-l_0)(l-l_1)(l-l_2)(l-l_3)\frac{f^{(4)}(\xi)}{4!} \tag{13}$$

where $\xi \in (a, b)$.

4 Correction

There are two prediction methods used in curve trajectory generation procedures: arc-length prediction and parameter prediction. In the arc-length prediction method, if the speeds or accelerations calculated through the prediction are greater than the maximum values, we modify the predicted values by either reducing the arc-length step or increasing the trajectory time to meet the robot requirements. Reducing the arc-length step greatly increases the calculation time, thus the preferred method is to increase the trajectory time, Figure 2.

In the arc-length prediction procedure, if the pose or speed calculated by the predicted parameter exceeds the expected values, we correct by increasing the time above $t_{\min}$, the minimum trajectory time specified by the user. Increasing trajectory time corresponds to slowing the trajectory speed while keeping the arc-length unchanged.

The Jacobian $\mathbf{J}$, constant for a fixed set of joint values, relates the tool speed vector $\mathbf{V}$ to the corresponding joint speed vector $\dot{\mathbf{q}}$ by the equation $\mathbf{V} = \mathbf{J}\dot{\mathbf{q}}$. By increasing the trajectory time $t > t_{\min}$, while keeping the arc-length unchanged, we decrease the tool speed and joint speed proportionally at a given robot configuration. Newer servocards permit a varying trajectory time which makes this correction method possible.

For correcting parameter prediction errors, we built a correction model according to the previous length errors and parameter errors. Given the previous step length L_0, and the previous step parameter U_0, we can predict the moving distance l through the spline curve model which corresponds to the predicted parameter u. The length error corresponding to the predicted parameter error is $\varepsilon_l = L - l$.

Suppose that the function $U = f(L)$ represents the exact relationship between parameter U and arc-length L, then according to Taylor series theory,

$$\begin{aligned} f(L) &= f(l+\varepsilon_l) = f(l) + f'(l)\varepsilon_l \\ &\quad + \frac{1}{2!}f''(l)\varepsilon_l^2 + \cdots + \frac{1}{n!}f^{(n)}(l)\varepsilon_l^n + E_n(l) \end{aligned} \tag{14}$$

where the error term is

$$E_n(l) = \frac{1}{(n+1)!}f^{(n+1)}(\xi)\varepsilon_l^{n+1} \tag{15}$$

and ξ is a variable that lies between l and L.

Because the length error ε_l is a small number, we ignore the second and higher order terms of the equation (14), and obtain a first order estimate of the parameter ε_u:

$$\varepsilon_u = U - u = f(L) - f(l) = f'(l)\varepsilon_l \tag{16}$$

If we approximate $f'(l)$ by

$$f'(l) = \frac{f(l) - f(l-\Delta_l)}{\Delta_l} = \frac{\Delta_u}{\Delta_l}, \tag{17}$$

where $\Delta_l = l - L_0$, and $\Delta_u = u - U_0$, we obtain the first order correction model

$$\varepsilon_u = \frac{\Delta_u}{\Delta_l}\,\varepsilon_l \tag{18}$$

To obtain a more accurate estimate of the parameter error ε_u, we apply a second order correction model by ignoring the third and higher order terms of (14).

$$\varepsilon_u = U - u = f(L) - f(l) = f'(l)\varepsilon_l + \frac{1}{2}f''(l)\varepsilon_l^2 \tag{19}$$

If we let Δ_{u0} and Δ_{l0} represent changes of parameter and arc-length at the last step, we can use Δ_{u0}, Δ_{l0}, Δ_u, and Δ_l to approximate $f''(l)$ by

$$f''(l) = \frac{f'(l) - f'(l - \Delta_l)}{\Delta_l} = \frac{\Delta_u\Delta_{l0} - \Delta_{u0}\Delta_l}{\Delta_l^2\Delta_{l0}} \tag{20}$$

The second order correction model then becomes

$$\varepsilon_u = \frac{\Delta}{\Delta_l}\,\varepsilon_l + \frac{\Delta_u\Delta_{l0} - \Delta_{u0}\Delta_l}{2\Delta_l^2\Delta_{l0}}\,\varepsilon_l^2 \tag{21}$$

5 Frame interpolation

Once a value of u is found, it can then be substituted back into the original NURBS or other parametric equations (equations (33)–(48) in Appendix) to find the Cartesian coordinates of the point along the curve. The trajectory planning procedures must generate not only the position of the tool frame, but the orientation of that frame as well.

The orientation is obtained by interpolating between the initial and final frames of the curve as a function of the arc-length. One way to define the frame axes is to assume the x axis coincident with the curve's tangent vector, and the z axis coincident with the normal vector of the curve (if 3-D curve, the normal vector will be coincident with the tool direction), while the third axis (y axis) is determined as the cross product of the x and z axes. If the curve lies on a surface, the x axis is defined as the tangent vector of the surface, the z axis normal to the surface, and the y axis is defined as the cross product of x and z axes – see equations (29)–(30) in the Appendix. This definition is useful for machining a space curve or sculptured surface which requires the cutter tool direction normal to the surface.

Consider the tool motion path segment in Figure 3

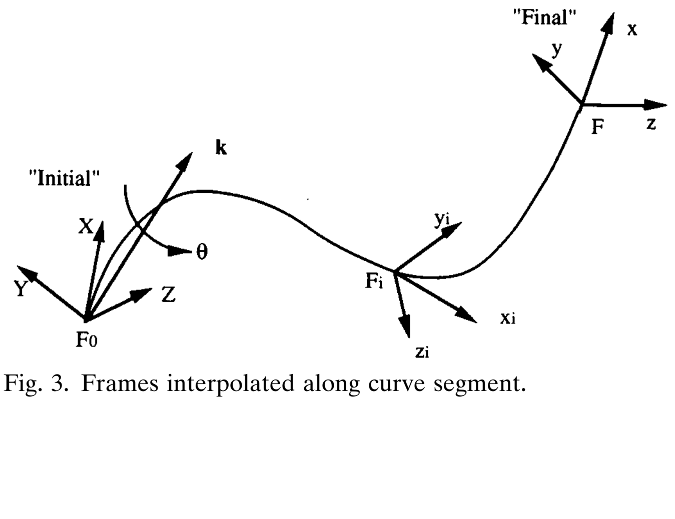

Fig. 3. Frames interpolated along curve segment.

with initial frame $\mathbf{F}_0$, final frame $\mathbf{F}$, and $\mathbf{F}_i$, an intermediate interpolated frame. Described as homogeneous transformations, these frames are characterized by a 3×3 rotational submatrix $\mathbf{R}$ and a 3×1 translation vector $\mathbf{P}$ which contains the position components of the frame origin with respect to a reference frame. Using $\mathbf{R}$, any orientation can be described by a screw angle θ about screw vector $\mathbf{k}$.

Several tool TCF (Tool Control Frame) interpolation types have been implemented (see Yang[14] for more detail):

FIXED_ORIENT – assumes that the TCF orientation is to be held constant during motion of the robot; thus, $\theta = 0$. The preferred tool motion type for gantry X-Y-Z robots and machine tools that have no orientation joints.

Z_POSE – assumes that the frames are arranged relative to the mechanisms such that the mechanism tool Z axis can be aligned with a target Z axis. The interpolation between two poses is made in two steps: 1) first, determine a vector normal to the TCF and target Z axes and then rotate about this vector from the initial TCF Z axis orientation to the target Z axis orientation; 2) next, rotate about the tool Z axis to align the tool frame X-Y axes with the target frame X-Y axes.

Z_POSE_NO_SPIN – same as Z_POSE, except that the spin about the tool Z axis is overriden. This would be the correct setting if the robot is used in surface polishing for example where the tools are oriented in a normal orientation relative to the surface.

FULL_POSE - requires the mechanism to place the tool frame at the same position and orientation of a target frame and should only be used when the mechanism has three orientation joints. Interpolation between two poses uses a screw vector and a translation to interpolate a tool frame from its initial orientation to its final target orientation by determining the orientation of the final frame relative to the initial frame. Using this frame the screw vector and screw angle are determined. Then the interpolated rotational frame is determined by first calculating a screw matrix determined by a rotation proportional to the distance moved along the curve $\mathbf{R}_i = \mathrm{Rot}(\mathbf{k}, \theta_i)$, where $\theta_i = \theta^* l/L$.

Next, we determine the translational matrix $\mathbf{T}_i$ which locates the interpolated frame origin. Multiplying, we determine the interpolated frame $\mathbf{F}_i$ as

$$\mathbf{F}_i = \mathbf{F}_0\mathbf{R}_i\mathbf{T}_i \tag{22}$$

X_TANGENT_POSE - similar to FULL_POSE by requiring the mechanism to place the tool frame at the pose of a target frame, but, in addition, X_TANGENT_POSE requires that the x-axis of each target frame coincide with the path tangent axis, and the z-axis be normal to the surface containing the curve. The process of determining intermediate frames along the tool path is to rotate the initial frame $\mathbf{X}$ axis into coincidence with x_i. Axis $\mathbf{k}$ is determined normal to X and x_i, and the θ_1 is the angle between X and x_i. The next step is to roll by angle θ_1 about axis x_i to align the z axes by an intermediate roll angle θ_2.

The intermediate frame $\mathbf{F}_i$, constructed from these two rotational operations, the origin translation, and initial frame $\mathbf{F}_0$ is

$$\mathbf{F}_i = \mathbf{F}_0\mathbf{R}_1(\mathbf{k}, \theta_1)\mathbf{R}_2(\mathbf{x}_i, \theta_2)\mathbf{T}_i \tag{23}$$

6 Inverse kinematics

Given the interpolated frame in Cartesian space, inverse kinematics are used to determine the joint values, speeds, and accelerations. However, the inverse of the Jacobian matrix is difficult to obtain, and the formulations vary for different kinds of robots. The simplest method of estimating joint speeds and accelerations is to divide the differences between two successive joint displacements and speeds by the trajectory time. This method is simple, and a fairly good approximation of the joint speeds and accelerations when the trajectory step is very small, but performs poorly for changes in joint direction.

An alternative method uses three trajectory steps to use a quadratic polynomial to approximate the joints speeds and accelerations. In this method we assume the joint accelerations constant, and let joint displacement q be a quadratic function of trajectory time:

$$q = b + c(t - t_i) + e(t - t_i)^2 \tag{24}$$

where b, c, and e are coefficients, and t is the current time. Joint speeds and accelerations are obtained from the first and second time derivatives of (24). It is assumed that q_{i-1}, q_i, and q_{i+1} are the displacements of the joint for the three most current trajectory steps, and t_{i-1}, t_i, and t_{i+1} are the corresponding trajectory times. Then we can obtain

$$b = q_i \tag{25a}$$

$$e = (q_{i+1}t_l + q_{i-1}t_c - q_i(t_l + t_c))/(t_lt_c(t_l + t_c)) \tag{25b}$$

$$c = (q_{i+1} - q_i - et_c^2)/t_c \tag{25c}$$

where $t_l = t_i - t_{i-1}$ represents the last trajectory step, and $t_c = t_{i+1} - t_i$ is the current trajectory step.

The joint speed and acceleration corresponding to the joint displacement q_{i+1} are then $\dot{q}_{i+1} = c + 2et_c$ and $\ddot{q}_{i+1} = 2e$. The newer servocards use high-order polynomials to blend position-velocity-time (PVT) moves in joint space. For these servocards, the quadratic approximation is generally sufficient. Yang[14] also considers a cubic approximation for acceleration continuity.

ERROR ANALYSIS AND TIME BOUNDING

On-line Cartesian trajectory control requires that all algorithmic calculations must be completed in a certain time, and that all errors be bounded by some specified tolerance. What is also important is the computational requirements above that required for normal trajectory control of motion along lines and circular arcs.

1 Error analysis

To analyze the trajectory errors, we introduce the following theorem. The proof can be found in Crampin.[15]

Theorem – Let $\mathbf{r}(u)$ be a twice continuously differentiable curve with a maximum curvature $k_{\max} \leq 1/\delta$, where $\delta > 0$. $\mathbf{P}_1$, $\mathbf{P}_2$ are two points on the curve $\mathbf{r}(u)$. If the curve length L_r from $\mathbf{P}_1$ to $\mathbf{P}_2$ along the curve $\mathbf{r}(u)$ satisfies

$$L_r \leq \pi/k_{\max} \tag{26}$$

$$\|\mathbf{P}_2 - \mathbf{P}_1\| \leq 2\left[\delta\left(\frac{2}{k_{\max}} - \delta\right)\right]^{1/2}, \tag{27}$$

then the error ε in replacing $\mathbf{r}(u)$ by the straight line P_1P_2 cannot exceed δ, Figure 4.

The path tolerance, measured by the maximum perpendicular distance between the path segment and the line connecting the two end points, is transformed into two simple inequalities in (26) and (27). It is clear that to interpolate a spatial curve accurately, more via points should be given in vicinities of large curvature segments.

By maintaining these inequalities, the errors generated by the methods will be within the error bound δ. We can either reduce the trajectory step length L_r for the large curvature segments, or modify the curve by reducing the maximum curvature $k_{\max}$. For a complex curve and tolerance δ, we first estimate the maximum curvature $k_{\max}$ for every estimated path segment using equation (32) in the Appendix. If the path length of a path segment satisfies the condition $L_r \leq \pi/k_{\max}$ and the inequality in equation (27) holds, the position errors of this estimated path segment are within the desired error bound; otherwise, we have to reduce the curve length steps, enlarge the specified tolerance or inform the user that the curvature is too large. From a motion viewpoint, we must either reduce the path speed or trajectory time.

For example, if we specify 150 mm/s as the tool speed, 75 Hz as the trajectory rate, and 0.05 mm as the specified tolerance, then the maximum curvature along the curve must be less than 0.0998 (radius of curvature = 10.0251 mm). If we use 50 mm/s as the tool speed for NC machine tool, 100 Hz as the trajectory rate, and 0.02 mm as the specified tolerance, then the maximum curvature along the curve must be less than 0.6359 (radius of curvature = 1.5725 mm). Since this is a conservative estimate, the actual error is less than the specified tolerance.

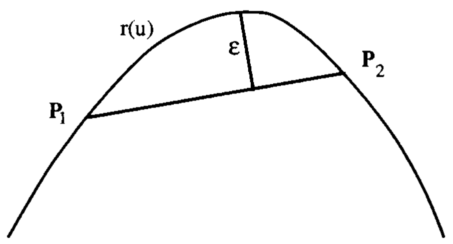

Fig. 4. Complex curve with error bound.

2 Time analysis

The on-line Cartesian trajectory algorithms outlined previously use, at most, one correction iteration and thus are bounded in time. The initialization stage applies a Gaussian quadrature method using tabulated data to approximate the integral. If we choose n-point Gaussian quadrature method to approximate the arc-length, the algorithm includes n function evaluations, n multiplications and $n - 1$ additions.

The length prediction procedures calculate the speed, acceleration time, and distance which are normal computations expected for trajectory generators. The relationship of the length change to the parameter change are additional computations which use the three point Gaussian quadrature method. This includes three function evaluations, three multiplications, and two additions.

The parameter prediction procedure uses quadratic or cubic extrapolation method to predict the parameter changes corresponding to the arc-length changes. If we use the quadratic extrapolation method (10), there are 12 multiplications/divisions and 14 additions/subtractions. If we use the cubic extrapolation method, there are 32 multiplications/divisions and 27 additions/subtractions.

The correction procedure uses simple first order models to correct the prediction errors. The first order model (18) includes two multiplications or divisions and two subtractions. The second order model (21) includes 10 multiplications or divisions and 6 additions or subtractions.

The frame interpolation procedure is more complex than the above procedures which includes matrix multiplication, vector dot and cross product operations, and multiplications, divisions, additions, and subtractions. However, all these operations are normal to trajectory generators. Similarly, inverse kinematics implement normal procedures.

As Table II demonstrates, only three stages require additional calculations as compared with the current Robline trajectory generator (typical of the generators found in many modern mechanism controllers). Therefore, the algorithms and procedures are time bounded.

SIMULATION AND EXPERIMENTAL RESULTS

The models, methods and algorithms developed in this research have been implemented in the C language and integrated with the Robline system. Because the Robline system is an open system, it allows users to implement their own trajectory generation functions to generate the trajectories. To test the developed methods and algorithms the Robline system was used to build the robot models and NURBS curves. Robpac processes were then used to specify motion parameters such as tool velocity and trajectory rates for moving along the trajectories. Simulation and physical experiments were conducted on several NURBS models. Only a few of the test cases are presented here. Again, refer to Yang[14] for details.

The first example moves the robot at a constant speed of 40 mm/s along a quadratic NURBS representation of a circle. The second example moves the robot along cubic NURBS curves which are a cross-section of a Blisk fan turbine blade. This example also illustrates the details of the trajectory procedures through three trajectory steps. Finally, to demonstrate these methods when applied to a physical robot, we move a 6-axis GE-P60 robot along a complex cubic NURBS curve.

Simulation example 1 – The first example uses a quadratic NURBS curve to represent a circle with radius 200 mm as shown in Figure 5. The knots, weights, and control points of the curve are shown in Table III. The desired path speed is 40 mm/s and the accel/decel is set to 100 mm/s^2. The robot's maximum tool speed is about 100 mm/s and its maximum acceleration/deceleration capability is about 1500 mm/s^2.

The real length of the circle is $2\pi R = 1256.63706144$, where $R = 200$ is the radius. The calculated length of the quadratic NURBS circle by using the 10 point Gaussian quadrature algorithm is 1256.63706140, an error of 0.00000004 mm.

In the example the minimum trajectory time is specified as 0.03 s. The tool speed errors from Figure 6 are less than 0.0025% (0.0010 mm/s).

To illustrate the shape of the joint motion curves, Figure 7 shows the joint angles, speeds and accelerations for joint 3 only. The other joint motion curves are similar. All values fall well within expected speed and acceleration limits.

Simulation example 2 – The second example considers a set of four cubic NURBS curves which are obtained

Table II. Additional calculation steps compared with Robline.

Procedures	Additional steps
Initialization	Normal
Length prediction	3 function evaluations, 3 multiplications and 2 additions
Parameter prediction	32 multiplications and 27 additions
Correction	2 multiplications and 2 subtractions
Frame interpolation	Normal
Inverse kinematics	Normal

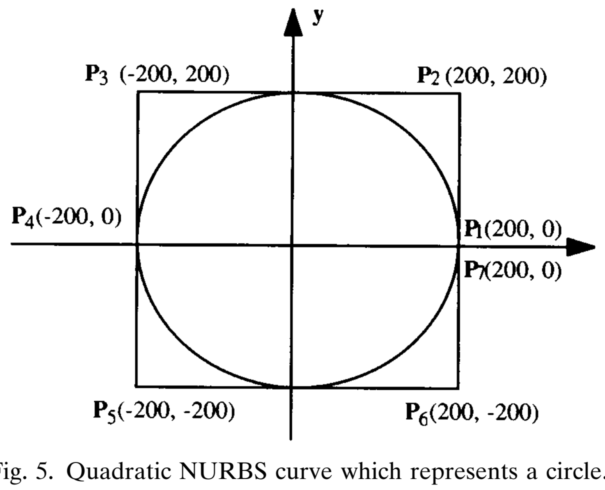

Fig. 5. Quadratic NURBS curve which represents a circle.

Table III. Quadratic NURBS curve parameters.

Knot vector	Weights	Control points
0.0	1.0	$\mathbf{P}_1(200, 0, 0)$
0.0	0.5	$\mathbf{P}_2(200, 200, 0)$
0.25	0.5	$\mathbf{P}_3(-200, 200, 0)$
0.5	1.0	$\mathbf{P}_4(-200, 0, 0)$
0.5	0.5	$\mathbf{P}_5(-200, -200, 0)$
0.75	0.5	$\mathbf{P}_6(200, -200, 0)$
1.0	1.0	$\mathbf{P}_7(200, 0, 0)$
1.0		

from a cross section of a turbine blade. The curve parameters for curve 2 are listed in Table IV. To demonstrate path following by the S100 robot the curves have been scaled up by a factor of 20.

We move the robot along all four curves in this simulation, but choose the second curve to illustrate the trajectory procedures. Table V lists three trajectory step results for the predictor-corrector method, where L and l represent the actual and predicted arc-length, u and U represent the predicted and corrected parameters, and ε_l and ε_u are the arc-length and parameter errors, respectively. We first calculate the trajectory length L through the velocity and acceleration profile and trajectory rates, then predict the parameter u using the predictor model, and find the corresponding arc-length l. The value $\varepsilon_1 = L - l$ is easy to obtain, and ε_u can be obtained through the corrector model. The corrected parameter U is finally found by $U = u + \varepsilon_u$. In this example, we specify 0.05 mm as the maximum tolerance of the curve, the three step maximum curvatures approximated are 0.1039, 0.1146, and 0.1244 respectively. The error check function passes the criteria set by the equations of (26) and (27). The maximum tool speed error is found to be 0.0287% (0.014 mm/s).

Experimental example 3 – To demonstrate these methods when applied to a physical robot, we move a 6-axis GE-P60 robot along the cubic NURBS curve shown in Figure 8. The speed and acceleration capabilities of this robot are similar to that of the S100 robot. The actual 2-D NURBS curve is shown in Figure 9.

One of the advantages of the Robline system is that

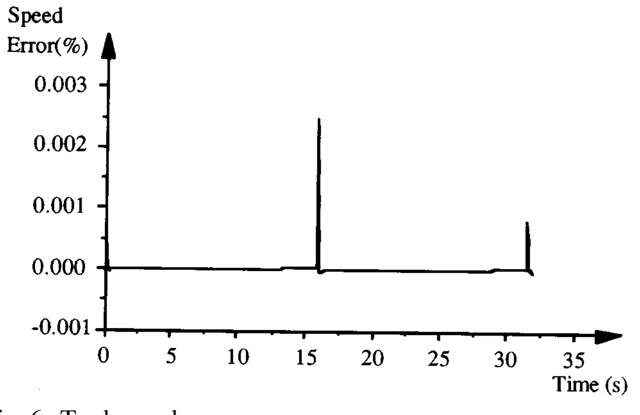

Fig. 6. Tool speed errors.

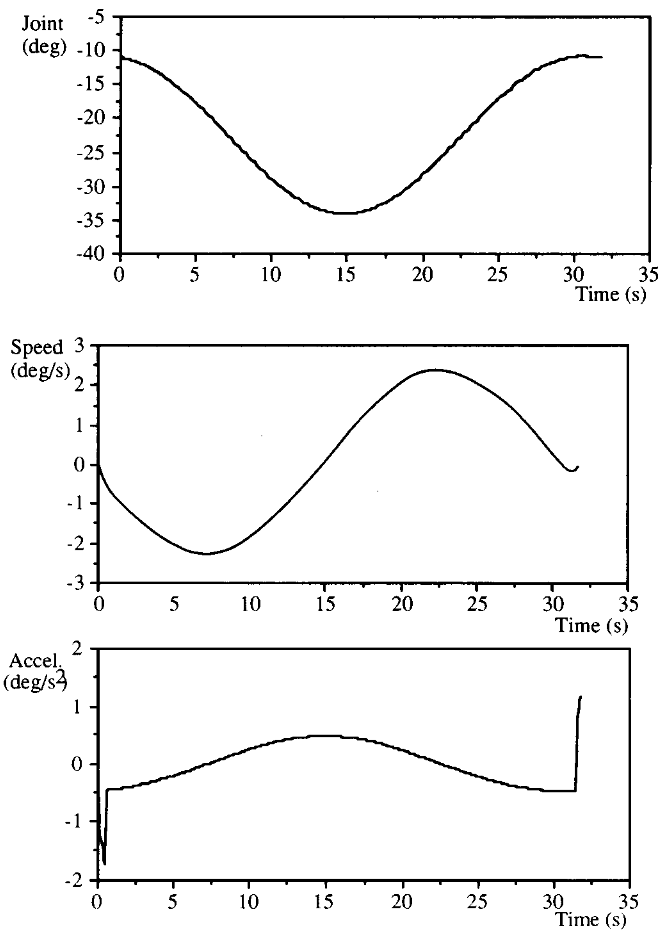

Fig. 7. Joint 3 angle, speed, and acceleration versus time.

it can drive both a simulated robot and an actual robot using the same control program. We first built the GE-P60 robot workcell model and the NURBS curve using a HP 735 workstation. We then wrote a simple Robpac process program to specify the necessary control parameters to simulate the motion control along the curve, thereby avoiding any possible collision of the robot.

The robot motion begins at rest and in the configuration represented by (−60, 0, 0, −20, 60, 0) degrees. The motion accelerates to the desired constant speed, and then comes to a full stop at the end of the trajectory. Again we use a simple trapezoidal velocity profile with a maximum tool tip speed of 100 mm/s and a constant acceleration/deceleration of 200 mm/s^2 to guide the tool speed changes. The quadratic interpolation

Table IV. Cubic NURBS curve 2 parameters.

Knot vector	Weights	Control points
0.0	1.0	$\mathbf{P}_1(1.3975, -0.1914, 4.6586)$
0.0	1.0	$\mathbf{P}_2(1.4054, -0.1904, 4.6620)$
0.0	1.0	$\mathbf{P}_3(1.4156, -0.1727, 4.6678)$
0.0244	1.0	$\mathbf{P}_4(1.4051, -0.1582, 4.6623)$
0.0492	1.0	$\mathbf{P}_5(1.3961, -0.1576, 4.6585)$
0.0492		
0.0492		

Table V. Three trajectory step results for predictor-corrector.

Step	l	L	ε_1	u	U	ε_u
1	11.4892	11.5200	0.0308	0.022052	0.022112	0.000060
2	12.9607	12.9991	0.0384	0.024976	0.025053	0.000077
3	14.4114	14.4413	0.0299	0.027920	0.027980	0.000060

method is applied to approximate the joint speeds and accelerations.

To drive the physical robot, we transferred the same robot models and control routines to the CIMETRIX OAC controller to move the GE-P60 along the curve. The controller CPU is an Intel/486-based PC computer operating at 50 MHz. The operating system is a realtime, multi-tasking Lynx system. CPU time is shared by several tasks: supervisor, trajectory planning, servo control (which has the highest priority), and the X-window manager. A 25 Hz trajectory rate was specified. The motions of the robot are smooth and continuous along the pre-defined curve geometry.

To compare the actual joint response, we wrote a simple program to gather actuator joint displacements from the PMAC servocard used in the controller. The differences between the calculated joint displacements and the actuator joint displacements which are fed back from the actual encoder are shown in Figure 10 for joint 3 (other joints perform similarly).

The maximum error between the specified joint displacements and actuator joint displacements through the controller is about 1 degree. These errors result from the robot following errors which are consistent with those obtained when driving the robot along linear and circular paths. Properly tuning the robot will dramatically reduce the robot following errors.

We also used the GE-P60 robot to verify the actual motion of the first and second simulation examples introduced in last section. Successful control of the robot demonstrated the capabilities of this research for physical control of the trajectory along a variety of Cartesian NURBS curves. A new CIMETRIX OAC controller based on a Pentium CPU will increase the trajectory rate to about 100 Hz. The algorithms will be able to achieve trajectory rates in excess of 100 Hz on NC machines since their inverse kinematics computations are simpler.

CONCLUSIONS

The algorithms and procedures developed in this research are time bounded (fixed number of calculation steps), and the trajectory errors meet certain tolerances as long as the maximum curvature of complex curve is less than a limiting value or the tool speed is less than a limiting value. Error analysis has demonstrated that the methods satisfy the typical tolerances required in robot motion and in NC machining.

To simulate the feasibility of the methods, several examples were considered. The first example used a NURBS curve to represent an exact circle. Moving along the curve at 40 mm/s, the maximum error of the calculated tool speeds was only 0.0025%, and the maximum joint acceleration of any joint experienced in the motion processes was less than 25 degrees/s^2, far below the maximum allowable joint acceleration.

Successful physical control of a GE-P60 robot along a Cartesian NURBS curve showed that the motion control system developed in this research could control mechanisms running in real-time. Multiple tests demonstrated that the 486 based PC controller could control a 6-axis GE-P60 robot (all revolute joints) moving along a 3-D NURBS curve at a trajectory rate of 30 Hz or less. A new CIMETRIX OAC controller based on a Pentium CPU will increase the trajectory rate to 100 Hz. The algorithms will achieve trajectory rates in excess of

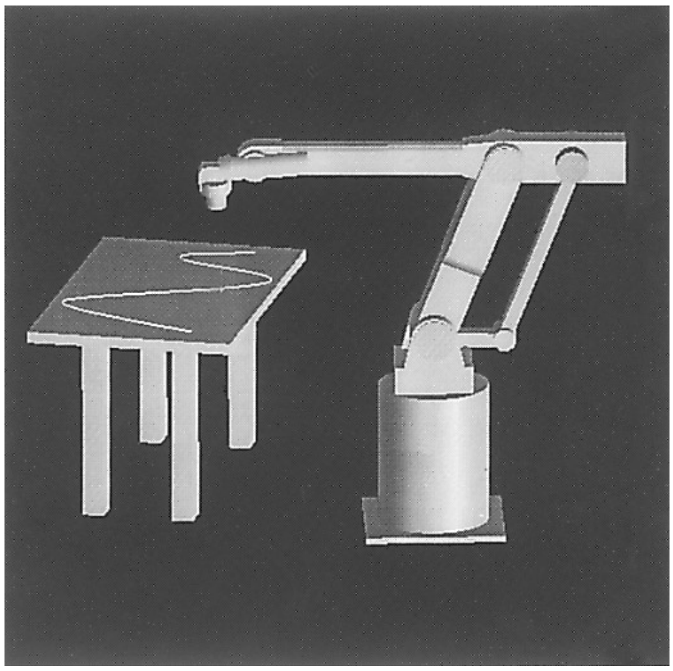
Fig. 8. GE-P60 Robot and NURBS curve.

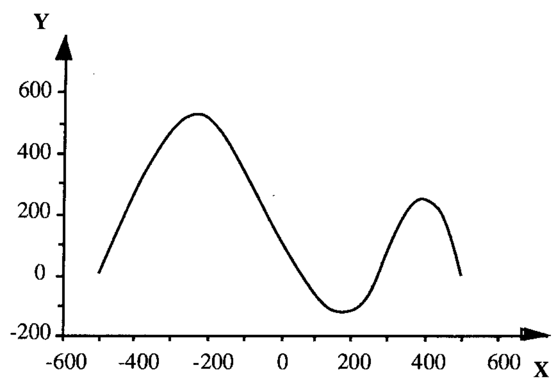

Fig. 9. 2-D cubic NURBS curve.

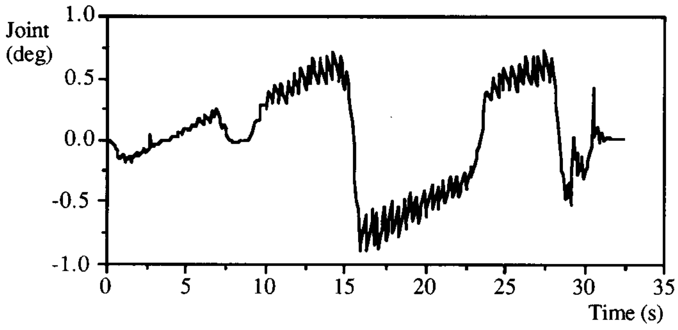

Fig. 10. Difference between actuator and joint 3 displacements.

100 Hz on NC machines since their inverse kinematics computations are simpler. The concepts and methodologies developed in this research are independent of the mechanisms being controlled.

The procedures are distinguished from most curve trajectory generation algorithms which transform a sequence of points into sets of joint displacements and approximate the Cartesian trajectory at the joint level. These new trajectory methods work directly in Cartesian space.

References

1. R. Paul, "Manipulator Cartesian Path Control" *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9,** No. 11, 702–711 (Nov., 1979.)
2. R.H. Taylor, "Planning and Execution of Straight Line Manipulator Trajectories" *IBM Journal of Research and Development* **23,** No. 4, 424–436 (July, 1979).
3. C.S. Lin and C. Po-Rong, "Joint Trajectories of Mechanical Manipulators for Cartesian Path Approximation" *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-13,** No. 6, 1094–1102 (Nov./Dec., 1983).
4. S.E. Thompson and V.P. Rajnikant, "Formulation of Joint Trajectories for Industrial Robots Using B-Spline" *IEEE Transactions on Industrial Electronics* **1E-34,** No. 2, 192–199 (May, 1987).
5. Y.H. Chang, T.T. Lee and C.H. Liu, "On-Line Approximate Cartesian Path Trajectory Planning for Robotic Manipulators" *IEEE Transactions on Systems, Man, and Cybernetics* **22,** No. 3, 542–547 (1992).
6. L.V. Aken and H. Van Brussel, "On-Line Robot Trajectory Control in Joint Coordinates by Means of Imposed Acceleration Profiles" *Proceedings of the 15th International Symposium on Industrial Robots* (September, 1985) pp. 1003–1008.
7. C. Froissart and P. Mechler, "On-line Polynomial Path Planning in Cartesian Space for Robot Manipulators" *Robotica* **11,** part 3, 245–251 (1993).
8. V. Beazel and E. Red, "Inaccuracy Compensation and Piecewise Circular Approximation of Parametric Paths" *Robotica* **11,** part 5, 413–425 (1993).
9. B.K. Choi, *Surface Modeling for CAD/CAM* (Elsevier Science Publishers, Amsterdam, 1991).
10. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design* (Academic Press, New York, 1990).
11. M.E. Mortenson, *Geometric Modeling* (John Wiley & Sons, New York, 1985).
12. J.H. Mathews, *Numerical Methods* (Prentice-Hall, Englewood Cliffs, 1987).
13. A.H. Stroud and D. Secrest, *Gaussian Quadrature Formula* (Prentice-Hall, Englewood Cliffs, 1966).
14. Z. Yang, "On-Line Cartesian Trajectory Control of Mechanisms along Complex Curves" *PhD Dissertation* (Brigham Young University, September, 1995).
15. M. Crampin, R. Guifo and G.A. Read, "Linear Approximation of Curve with Bounded Curvature and a Data Reduction Algorithm" *Computer-Aided Design* **17,** No. 6, 257–261 (1985).

APPENDIX – REVIEW OF COMPLEX CURVES

1. Curve geometry

A space curve is conveniently represented by a parametric vector equation of the form $\mathbf{r}(u) = [x(u), y(u), z(u)]$. The derivative of $\mathbf{r}(u)$ becomes $\mathbf{r}'(u) = d\mathbf{r}(u)/du = [dx/du,\ dy/du,\ dz/du]$. Higher order derivatives are defined similarly.

Let s represent the arc length along curve $\mathbf{r}(u)$, then

$$s = \int_{u_0}^{u_1} \left| \frac{d\mathbf{r}(u)}{du} \right| du \tag{28}$$

Referencing Figure 11, the unit tangent vector for curve $\mathbf{r}(u)$ is defined as

$$\mathbf{T} = \frac{d\mathbf{r}}{ds} = \frac{\mathbf{r}'(u)}{|\mathbf{r}'(u)|} \tag{29}$$

By differentiating $\mathbf{T}$ with respect to u ($\mathbf{T}' = d\mathbf{T}/du$) and normalizing, we obtain the principal normal vector $\mathbf{N}$ of curve $\mathbf{r}(u)$, which is orthogonal to $\mathbf{T}$:

$$\mathbf{n} = \frac{\mathbf{T}'(u)}{|\mathbf{T}'(u)|} \tag{30}$$

A third vector perpendicular to both $\mathbf{T}$ and $\mathbf{N}$, called the binormal vector, is given by $\mathbf{B} = \mathbf{T} \times \mathbf{N}$. Frame $\{\mathbf{T}, \mathbf{N}, \mathbf{B}\}$ is called the Frenet frame.

The curvature k of $\mathbf{r}(u)$ is defined as

$$k = \left| \frac{d\mathbf{T}}{ds} \right| \tag{31}$$

By applying the chain rule, the curvature is obtained as

$$k = \frac{|\mathbf{r}' \times \mathbf{r}''|}{|\mathbf{r}'|^3} \tag{32}$$

where $\mathbf{r}' = d\mathbf{r}/du$ and $\mathbf{r}'' = d\mathbf{r}'/du$.

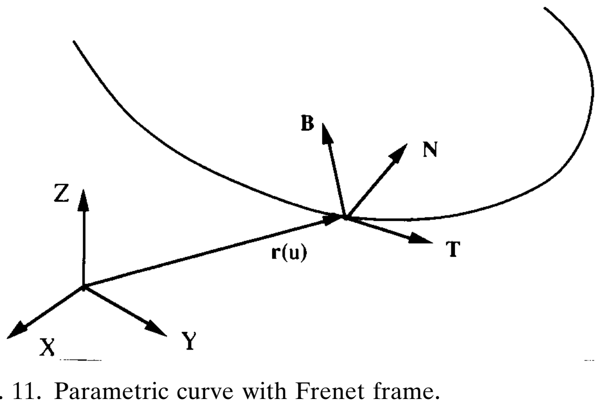

Fig. 11. Parametric curve with Frenet frame.

2. Bezier curve
A degree n Bezier curve with $n+1$ control points $\{\mathbf{P}_i\colon i=0,1,\ldots,n\}$ is defined as

$$\mathbf{P}(u)=\sum_{i=0}^{n} B_i^n(u)\mathbf{P}_i \quad (0\le u\le 1) \tag{33}$$

where u is the parameter, and $B_i^n(u)$ is the blending function called the Bernstein polynomial:

$$B_i^n(u)=\binom{n}{i}(1-u)^{n-i}u^i \quad (i=0,1,\ldots,n) \tag{34}$$

where $\binom{n}{i}=\dfrac{n!}{(n-i)!\,i!}$.

The first derivative of the Bernstein polynomial (34) is evaluated as

$$\begin{aligned}\frac{d}{du}B_i^n(u)&=\frac{d}{du}\left(\frac{n!}{(n-i)!\,i!}u^i(1-u)^{n-i}\right)\\&=n(B_{i-1}^{n-1}(u)-B_i^{n-1}(u))\end{aligned} \tag{35}$$

Thus, the derivative of a Bezier curve of degree n is obtained as follows:

$$\begin{aligned}\frac{d}{du}\mathbf{P}(u)&=\frac{d}{du}\left(\sum_{i=0}^{n}B_i^n(u)\mathbf{P}_i\right)\\&=\sum_{i=0}^{n}n(B_{i-1}^{n-1}(u)-B_i^{n-1}(u))\mathbf{P}_i\\&=n\sum_{i=0}^{n-1}B_i^{n-1}(u)(\mathbf{P}_{i+1}-\mathbf{P}_i)\end{aligned} \tag{36}$$

The second derivative of a Bezier curve is also easily obtained as

$$\begin{aligned}\frac{d^2}{du^2}\mathbf{P}(u)&=\frac{d}{du}\left(\frac{d}{du}\mathbf{P}(u)\right)\\&=n\sum_{i=0}^{n-1}(\mathbf{P}_{i+1}-\mathbf{P}_i)\frac{d}{du}B_i^{n-1}(u)\\&=n(n-1)\sum_{i=0}^{n-2}(\mathbf{P}_{i+2}-2\mathbf{P}_{i+1}+\mathbf{P}_i)B_i^{n-2}(u)\end{aligned} \tag{37}$$

3. Rational Bezier curve
A degree n rational Bezier curve is defined as

$$\mathbf{R}(u)=\frac{\sum_{i=0}^{n}\mathbf{P}_iw_iB_i^n(u)}{\sum_{i=0}^{n}w_iB_i^n(u)} \quad (0\le u\le 1) \tag{38}$$

where $\{\mathbf{P}_i\colon i=0,1,\ldots,n\}$ are control vertices, $\{w_i\colon i=0,1,\ldots,n\}$ are weights, and $B_i^n(u)$ is the Bernstein polynomial.

A rational curve model provides more degrees of freedom in defining curve shape. If we increase one weight w_i, the Bezier curve is pulled toward the corresponding vertex $\mathbf{P}_i$. If all weights $\{w_i\}$ are equal to 1, the rational Bezier curve becomes the ordinary Bezier curve. The rational Bezier curve can be expressed as

$$\mathbf{R}(u)=\frac{\sum_{i=0}^{n}\mathbf{P}_iw_iB_i^n(u)}{\sum_{i=0}^{n}w_iB_i^n(u)}=\frac{\mathbf{P}(u)}{w(u)} \quad (0\le u\le 1) \tag{39}$$

The derivative of a rational Bezier curve is obtained by differentiating (39) with respect to u:

$$\frac{d}{du}\mathbf{R}(u)=\frac{w(u)\dfrac{d}{du}\mathbf{P}(u)-\mathbf{P}(u)\dfrac{d}{du}w(u)}{w^2(u)} \tag{40}$$

By differentiating (40), the second derivative of a rational Bezier curve becomes

$$\begin{aligned}\frac{d^2}{du^2}\mathbf{R}(u)=&\frac{w(u)\dfrac{d^2}{du^2}\mathbf{P}(u)-\mathbf{P}(u)\dfrac{d^2}{du^2}w(u)}{w^2(u)}\\&-\frac{2\dfrac{d}{du}w(u)}{w(u)}\frac{d}{du}\mathbf{R}(u)\end{aligned} \tag{41}$$

From the previous equations and (36) and (37), we evaluate the first and second derivatives at the initial point of a rational Bezier curve:

$$\frac{d}{du}\mathbf{R}(0)=n\frac{w_1}{w_0}(\mathbf{P}_1-\mathbf{P}_0) \tag{42}$$

$$\begin{aligned}\frac{d^2}{du^2}\mathbf{R}(0)=&\,n(n-1)\frac{w_2}{w_0}(\mathbf{P}_2-\mathbf{P}_0)\\&+2\left(1-n\frac{w_1}{w_0}\right)\frac{d}{du}\mathbf{R}(0)\end{aligned} \tag{43}$$

4. B-spline curve
For a given sequence of 3D control points $\{\mathbf{P}_i\}$ $i=0,1,\ldots,n,$ and a non-decreasing knot sequence $(t_0,t_1,\ldots,t_{n+k-2})$, a B-spline curve of degree $k-1$ is defined as

$$\mathbf{P}(u)=\sum_{i=0}^{i=n}\mathbf{P}_iN_{i,k}(u) \tag{44}$$

where u is the parameter, and $N_{i,k}(u)$ is the B-spline basis function. The B-spline basis functions are defined recursively by the following expressions:

$$\begin{aligned}N_{i,1}(u)&=1,\quad \text{if } u\in[t_i,t_{i+1}]\\&\quad\ 0,\quad \text{otherwise}\end{aligned} \tag{45}$$

$$\begin{aligned}N_{i,k}(u)=&\frac{(u-t_i)N_{i,k-1}(u)}{t_{i+k-1}-t_i}\\&+\frac{(t_{i+k}-u)N_{i+1,k-1}(u)}{t_{i+k}-t_{i+1}}\end{aligned} \tag{46}$$

for $i=0,1,\ldots,n,$ where k controls the degree $(k-1)$ of the resulting polynomials in u and thus controls the continuity of the curve. This function is known as the Cox-de Boor recursive function.

5. Non-Uniform Rational B-Spline (NURBS)

To define a rational B-spline, we make use of the homogeneous coordinate. If $\mathbf{P} = (x, y, z)$ is a point in 3-D Euclidean space, we denote a corresponding point in 4-D homogeneous space by $\mathbf{H} = [wx, wy, wz, w]$, where $w > 0$. We call w the homogeneous coordinate.

We define a polynomial B-spline curve in homogeneous space by the vector equation

$$\begin{aligned}\mathbf{P}_h(u) &= (x(u), y(u), z(u), w(u)) \\ &= \sum_{i=0}^{i=n} \mathbf{H}_i N_{i,k}(u) \qquad (47)\end{aligned}$$

where the $N_{i,k}(u)$ are the usual kth-order polynomial B-spline basis functions, $\mathbf{H}_i = (w_i x_i, w_i y_i, w_i z_i, w_i)$ are the control points in homogeneous space, and the knot vector $\{t_i\colon i = 0, 1, \ldots, n + k - 2\}$ is the same as defined previously.

$\mathbf{P}_h(u)$ forms a set of points in 4-D homogeneous space. The B-spline $\mathbf{P}(u)$ projection in 3-D space is obtained by dividing the first three coordinates of each point by its homogeneous coordinate. $\mathbf{P}(u)$ is called the rational B-spline curve and is defined by

$$\mathbf{P}(u) = \frac{\sum_{i=0}^{i=n} w_i \mathbf{P}_i N_{i,k}(u)}{\sum_{i=0}^{i=n} w_i N_{i,k}(u)} \qquad (48)$$

where $\{w_i\colon i = 0, 1, \ldots, n\}$ are the weights.

To evaluate a rational B-spline curve at a parameter value u, we may apply the de Boor algorithm or the matrix form[9] to both the numerator and denominator of (48), and finally divide through. This corresponds to the evaluation of a 4-D non-rational curve with control vertices $\{w_i x_i, w_i y_i, w_i z_i, w_i\}^T$ and to projecting the result into 3-D space. To evaluate the derivatives of NURBS curve, convert the NURBS curve into a rational Bezier curve to calculate the derivatives as shown previously in the Appendix. The NURBS curve can represent almost all curves if we properly plan the control points, knot vectors, and weights.