The Dissertation Committee for Peter Setterlund March Certifies that this is the approved version of the following dissertation:

Geometric-based Spatial Path Planning

Committee:

Delbert Tesar, Supervisor

Chandrajit Bajaj

Ronald Barr

Richard Crawford

Chetan Kapoor

Geometric-based Spatial Path Planning

by

Peter Setterlund March, B.S.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2008

Dedication

I dedicate this work to my parents for always encouraging me to pursue difficult goals, and to Samantha for all of her support.

Acknowledgements

I would like to thank Dr. Tesar, Dr. Kapoor, and Dr. Pryor for all of their help and guidance over the course of my research. The opportunities and resources provided to me over the course of my time at the Robotics Research Group have given me a rich and rewarding graduate student experience as well as preparing me for future challenges.

Geometric-based Spatial Path Planning

Publication No._____

Peter Setterlund March, Ph.D

The University of Texas, 2008

Supervisor: Delbert Tesar

Cartesian space path planning involves generating the position and orientation trajectories for a manipulator end-effector. Currently, much of the literature in motion planning for robotics concentrates on topics such as obstacle avoidance, dynamic optimizations, or high-level task planning. The focus of this research is on operator-generated motions. This will involve analytically studying the effects of higher-order properties (such as curvature and torsion) on the shape of spatial Cartesian curves. A particular emphasis will be placed on developing physical meanings and graphical visualization for these properties to aid the operator in generating geometrically complex motions.

This research begins with a brief introduction to the domain of robotics and manipulator motion planning. An overview of work in the area of manipulator motion planning will demonstrate a lack of research on generating geometrically complex spatial paths. To pursue this goal, this report will then provide a review of the theory of algrebraic curves and their higher-order properties. This involves an evaluation of several different representations for both planar and spatial curves. Then, a survey of interactive

curve generation techniques will be performed, which will draw from fields outside of robotics such as Computer Graphics and Computer-Aided Design (CAD).

In addition to the reviewed methods, a new method for describing and generating spatial curves is proposed and demonstrated. This method begins with the study of a finite set of local geometric motion shapes (circular arcs, cusps, helices, etc). The local geometric shapes are studied in terms of their geometric parameters (curvature and torsion), analyzed to give physical meaning to these parameters, and displayed graphically as a family of curves based on these controlling parameters. This leads to the development of path constraints with well-defined physical meaning. Then, a curve generation method is developed that can convert these geometric constraints into parametric constraints and blend between them to form a complete motion program (cycle) of smooth paths connecting several carefully developed local curve properties. Up to ten distinct local curve shapes were developed in detail and one curve cycle demonstrated how all this could be combined into a full path planning scenario. Finally, the developed methods are packaged together into existing software and applied to an example demonstration.

TABLE OF FIGURES

TABLE OF TABLES

1. CHAPTER ONE

Introduction

This research is contained in the broad domain of manipulator motion planning with an emphasis on the geometry of Cartesian space paths. Much of the past and current research in this area is focused on manipulator system topics such as obstacle avoidance, dynamic optimizations, and high-level task planning. However, little work has been done in studying and generating complex geometries for manipulator path plans. As the tasks required for robotic systems become increasingly complex, a better understanding of the geometry of spatial curves and how this relates to motion planning will be required. The goal of this work is to create for describing and generating geometrically complex paths with well understood physical meaning.

This chapter will first introduce the domain of robotics by describing basic geometry and modeling techniques for serial manipulators. Then, a review of common techniques for joint and Cartesian space planning will be provided with an emphasis on complex Cartesian motions. This review will demonstrate a need for a more geometric based approach to planning these motions. Finally, the last section of this chapter will discuss the structure of the rest of this work.

1.1. MANIPULATOR MODELING

In order to understand the mathematics behind motion planning, it is important to understand the geometry of serial robotic systems. Serial robotic manipulators are made up of a series of controllable variable joints. These joints are usually either prismatic (P) or revolute (R). A prismatic joint provides linear motion along an axis, and a revolute

joint provides rotation about an axis. The degrees of freedom (DOF) of a serial robot is defined to be the number of active variable joints (i.e. each joint is driven).

The current configuration of a serial system is a function of the current joint positions. This is shown in Equation (1.1) where $\{x, y, z, \psi_x, \psi_y, \psi_z\}$ represents the position and orientation of the manipulator end-effector. The joint position vector, $\{\phi_1, \phi_2, \cdots, \phi_n\}$, is the input to the system (where *n* is the degrees of freedom), and the manipulator end-effector position and orientation is the output.

$$\begin{aligned} x &= f(\phi_1, \phi_2, \cdots, \phi_n) \\ y &= f(\phi_1, \phi_2, \cdots, \phi_n) \\ z &= f(\phi_1, \phi_2, \cdots, \phi_n) \\ \psi_x &= f(\phi_1, \phi_2, \cdots, \phi_n) \\ \psi_y &= f(\phi_1, \phi_2, \cdots, \phi_n) \\ \psi_z &= f(\phi_1, \phi_2, \cdots, \phi_n) \end{aligned} \tag{1.1}$$

The joint position vector is usually defined as a function of time. This means that the joint velocities and accelerations can be easily obtained by differentiating, as shown in Equation (1.2).

$$\begin{aligned} \phi &= f(t) = \{\phi_1(t), \phi_2(t), \cdots, \phi_n(t)\} \\ \dot{\phi} &= d\phi/dt = \{\dot{\phi}_1(t), \dot{\phi}_2(t), \cdots, \dot{\phi}_n(t)\} \\ \ddot{\phi} &= d\dot{\phi}/dt = \{\ddot{\phi}_1(t), \ddot{\phi}_2(t), \cdots, \ddot{\phi}_n(t)\} \end{aligned} \tag{1.2}$$

1.1.1. DH Parameters

Denavit-Hartenberg (DH) parameters are one of the most common methods for describing the geometry of serial manipulators. This method was developed by Denavit and Hartenberg [13] in 1955. It involves defining a set of four parameters (a, α, d, θ) that represent the coordinate transformations from one joint axis to the next. In each of these transformations, some of the parameters are variable and others are fixed. For a revolute joint, the θ parameter is usually defined to be variable. For a prismatic joint, the *d*

parameter is usually defined to be variable. For a serial robot, the number of variable parameters will be equal to the DOF. A visual representation of these parameters is shown in Figure 1.1.

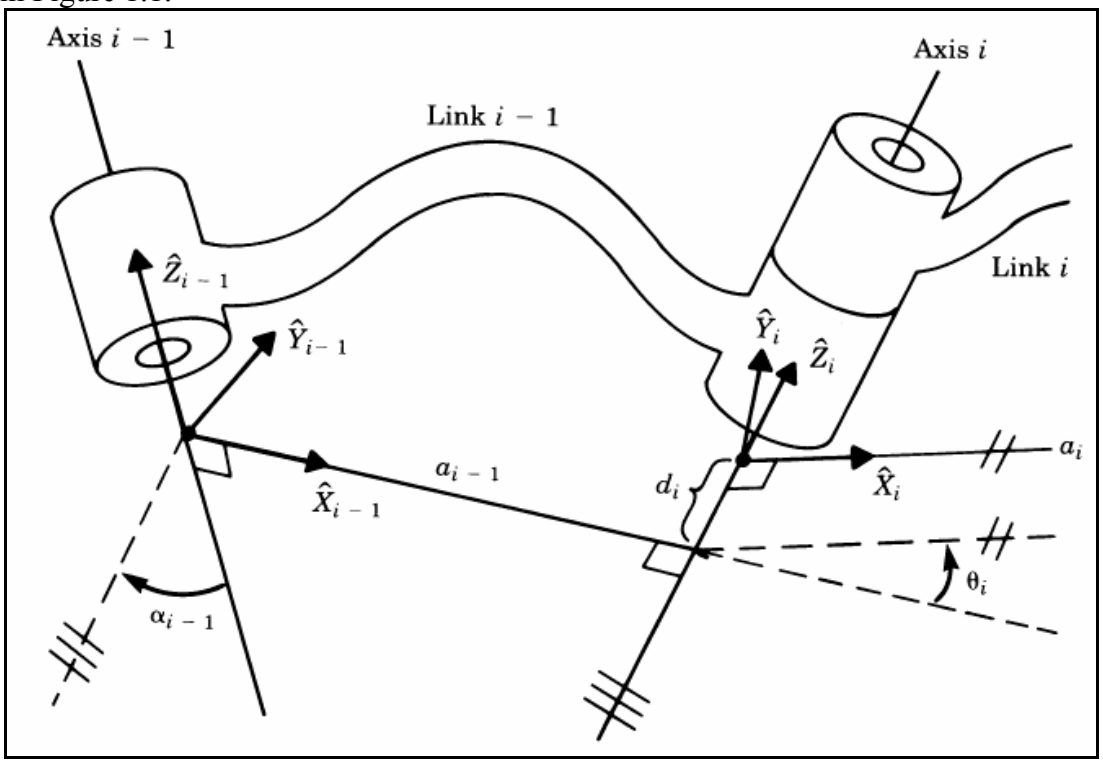


Figure 1.1 DH Parameters and link frames. Craig [13]

The DH Parameters are defined as [13]:

- a_i = the distance from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ measured along $\hat{X}_i$
- α_i = the angle between $\hat{Z}_i$ and $\hat{Z}_{i+1}$ measured about $\hat{X}_i$
- d_i = the distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ measured along $\hat{Z}_i$
- θ_i = the angle between $\hat{X}_{i-1}$ and $\hat{X}_i$ measured about $\hat{Z}_i$

Each transformation between consecutive joints is treated individually as a 4x4 transformation matrix. A transformation matrix represents a spatial transform (translation and rotation) between two coordinate frames. The transformation matrix [13]

to move from a frame attached to joint i to a frame attached to joint i-1 is shown in Equation (1.3).

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(1.3)} $$

The total transformation from the fixed base frame to the variable end-effector (EE) frame is the combination of the transformation matrices for each consecutive joint pair. For an n-DOF serial system, this is shown in Equation (1.4) where ${}^{0}_{n}T$ is the transformation from the base frame to the EE frame.

$$ {}^{0}_{n}T = \left[{}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\cdots{}^{n-1}_{n}T \right] \quad \textbf{(1.4)} $$

1.1.2. End-Effector Representations

The modeling method described in the previous section defines the location of the EE as a transformation from the base frame into the EE frame. This transformation matrix has the form shown in Equation (1.5). The parameter ${}^{A}P_{B}$ represents a 3x1 vector of the x, y, and z positions of the EE, and ${}^{A}_{B}R$ is a 3x3 rotation matrix that rotates the base frame into the tool frame. While this provides a good interpretation of the position, it is difficult to visualize and control the orientation of the EE from a 3x3 rotation matrix representation. This section will describe several other methods for describing spatial orientations.

$$ \begin{bmatrix} {}^{A}_{B}R & {}^{A}P_{B} \\ 0\ 0\ 0 & 1 \end{bmatrix} \quad \textbf{(1.5)} $$

1.1.2.1. Fixed XYZ Angles

Fixed XYZ angles are one of the most common methods for describing EE orientation. This representation will describe the orientation as a 3x1 vector of angles $\{\alpha, \beta, \gamma\}$. These angles represent three rotations around the fixed world frame (i.e. one rotation about the fixed x-axis, one rotation about the fixed y-axis, and one rotation about the fixed z-axis). While this leads to a compact description of the EE in a 3x1 vector, it is often difficult to visualize how a rotation around one of the fixed frame axes will change the EE when controlling a robot. Further information can be found in [13].

1.1.2.2. Euler Angles

Euler angles are similar to Fixed XYZ and are also represented in a 3x1 vector $\{\alpha, \beta, \gamma\}$. However, they are relative to the rotating frame. For example, Euler XYZ angles will represent a rotation about the fixed x-axis followed by a rotation around the new y-axis followed by a rotation about the new z-axis. Euler angles can be defined as rotations around any order of the *x*, *y*, and *z* axes as long as no two consecutive axes are the same. For example, Euler XYZ and Euler ZYZ are valid representations, but Euler ZZX is not. Euler angles suffer from many of the same problems as Fixed XYZ angles, but they allow for direct control over rotation about one of the EE frame's axes. For example, changing the γ value in an Euler XYZ representation will cause a rotation about the EE's z-axis. Fixed XYZ and Euler angles are often stored along with the position of the EE in a 6x1 vector called a handpose. Further information can be found in [13].

1.1.2.3. Equivalent Axis

Another common representation is the Equivalent-Axis. This defines the orientation as a rotation about a single axis in space. This representation consists of a 3x1

vector and an angle $R(\hat{\mathbf{n}},\theta)$. The rotation matrix to describe the orientation can be determined from Equation (1.6) [13]. Equivalent Axis rotations can be useful for interpolating between initial and final orientations, however; this representation is not very useful for visualizing an EE's orientation.

$$\begin{pmatrix} n_x n_x(1-\cos\theta)+\cos\theta & n_x n_y(1-\cos\theta)-n_z\sin\theta & n_x n_z(1-\cos\theta)+n_y\sin\theta \\ n_x n_y(1-\cos\theta)+n_z\sin\theta & n_y n_y(1-\cos\theta)+\cos\theta & n_y n_z(1-\cos\theta)-n_x\sin\theta \\ n_x n_z(1-\cos\theta)-ny\sin\theta & n_y n_z(1-\cos\theta)+n_x\sin\theta & n_z n_z(1-\cos\theta)+\sin\theta \end{pmatrix} \quad \textbf{(1.6)}$$

1.1.2.4. Quaternions

Quaternions are commonly used in industry to represent spatial orientations and are similar to Equivalent axes. A quaternion is a 4x1 vector $\left(q = w + x\hat{\mathrm{i}} + y\hat{\mathrm{j}} + z\hat{\mathrm{k}}\right)$ that contains a 3x1 vector (x, y, z) representing an axis in space and a scalar value (w) representing the cosine of the half-angle of rotation about this axis [41]. Quaternions, like equivalent axes, are often used for interpolation between two orientations. The Spherical Linear Interpolation (SLERP) method, shown in Equation (1.7), is the most common method used for interpolation. The θ in this equation is the angle between the initial and final quaternions given by $\cos^{-1}(q_0 \cdot q_1)$. This method provides a constant angular velocity with respect to the independent parameter t.

$$q(t) = \frac{q_0 \sin\left((1-t)\theta\right) + q_1 \sin(t\theta)}{\sin\theta}, t \in [0,1] \quad \textbf{(1.7)}$$

1.1.3. Kinematics Model

Thomas and Tesar [47] developed a kinematics model for serial manipulators based on generalized influence coefficients. These coefficients represent a direct geometric mapping between input and output parameters. This model is generalized and can be used on a wide variety of systems ranging from complex hyper-redundant

manipulators to simple planar systems. The next few sections will describe the analytic development of this model.

1.1.3.1. First Order Influence Coefficients

A generalized method for mapping the time derivatives of the input and output parameters of a manipulator was developed by Thomas and Tesar [51]. This method relies on defining kinematic influence coefficients that are functions only of the current geometry of the system. Let the output position vector be described by $\mathbf{P} = \{x, y, z, \psi_x, \psi_y, \psi_z\}$. The time derivative of this position is shown in Equation (1.8).

$$\dot{\mathbf{P}} = \frac{d\mathbf{P}}{dt} = \frac{\partial \mathbf{P}}{\partial \phi}\frac{d\phi}{dt} = \left[G_\phi^p\right]\dot{\phi} \tag{1.8}$$

The $\left[G_\phi^p\right]$ in this equation represents a matrix containing the first order influence coefficients, or G functions, relating the input joint velocities to the output EE velocities. Each individual G function maps a certain input to a certain output. For example, a G function mapping the effect of the first joint on the *x* position of the output would look like $\left[G_{\phi_1}^x\right] = \partial x / \partial \phi_1$. The entire matrix of G functions is shown in Equation (1.9). This matrix relates the input joint velocities to the manipulator's EE velocities. It is often referred to as the Jacobian matrix [13].

$$\left[G_\phi^p\right] = \begin{bmatrix} \frac{\partial x}{\partial \phi_1} & \frac{\partial x}{\partial \phi_2} & \cdots & \frac{\partial x}{\partial \phi_n} \\ \frac{\partial y}{\partial \phi_1} & \frac{\partial y}{\partial \phi_2} & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial \psi_z}{\partial \phi_1} & \cdots & \cdots & \frac{\partial \psi_z}{\partial \phi_n} \end{bmatrix} \tag{1.9}$$

1.1.3.2. Second Order Influence Coefficients

Similarly, the second order influence coefficients can be derived that relate the input accelerations to the output accelerations. Equation (1.10) shows the second derivative with respect to time of the output position vector, **P**.

$$\ddot{\mathbf{P}} = \frac{d^2\mathbf{P}}{dt^2} = \frac{d}{dt}\left(\frac{\partial \mathbf{P}}{\partial \phi}\frac{d\phi}{dt}\right) = \frac{\partial \mathbf{P}}{\partial \phi}\ddot{\phi} + \frac{\partial^2 \mathbf{P}}{\partial \phi^2}\dot{\phi}^2 \qquad \textbf{(1.10)}$$

It is interesting to note that the first order influence coefficient, $\frac{\partial \mathbf{P}}{\partial \phi}$, appears in this derivation. This term relates the effect of joint accelerations on output accelerations while the second term relates centripetal and Coriolis effects on the output accelerations. Equation (1.11) shows this equation in terms of influence coefficients, where $\left[H^p_{\phi\phi}\right] = \frac{\partial^2 \mathbf{P}}{d\phi^2}$.

$$\ddot{\mathbf{P}} = \left[G^p_\phi\right]\ddot{\phi} + \dot{\phi}^T\left[H^u_{\phi\phi}\right]\dot{\phi} \qquad \textbf{(1.11)}$$

The $\left[H^p_{\phi\phi}\right]$ represents the second order influence coefficients or H functions. In Equation (1.11), the H functions are grouped together into a tensor known as the Hessian. This tensor contains as many planes as the system output DOFs.

1.1.4. Dynamic Model

The influence coefficients derived in the previous section can be further expanded to create a generalized dynamic model of manipulators. This dynamic model will be briefly discussed here; a more detailed derivation can be found in Thomas and Tesar [51]. Equation (1.12) shows the formulation of the joint torques resulting from system inertia.

$$\tau^I_n = \left[I^*_{\phi\phi}\right]\ddot{\phi} + \dot{\phi}^T\left[P^*_{\phi\phi\phi}\right]\dot{\phi} \qquad \textbf{(1.12)}$$

The $\left[I^*_{\phi\phi}\right]$ term in this equation is the *effective inertia matrix* and represents the effects of mass and moment of inertia on each of the active reference joint parameters ϕ_i.

Its formulation is shown in Equation (1.13). The M_{jk} term represents the link centroidal masses, and the $\left[\prod^{jk}\right]$ represents the link rotary inertias.

$$\left[I^*_{\phi\phi}\right] = \sum_{j=1}^{N}\left\{M_{jk}\left[{}^{j}G^{c}_{\phi}\right]^{T}\left[{}^{j}G^{c}_{\phi}\right] + \left[G^{jk}_{\phi}\right]^{T}\left[\Pi^{jk}\right]\left[G^{jk}_{\phi}\right]\right\} \tag{1.13}$$

The $\left[P^*_{\phi\phi\phi}\right]$ in Equation (1.12) is the *inertial power array*. This includes centripetal and Coriolis effects from the inertia. The formulation for $\left[P^*_{\phi\phi\phi}\right]$ is shown in Equation (1.14).

$$\begin{aligned}\left[P^*_{\phi\phi\phi}\right] = \sum_{i=1}^{n}\Bigg\{ & M_{jk}\left(\left[{}^{j}G^{c}_{\phi}\right]\circ\left[{}^{j}H^{c}_{\phi\phi}\right]\right) \\ & +\left(\left(\left[{}^{j}G^{c}_{\phi}\right]\circ\left[\Pi^{jk}\right]\right)\circ\left(\left[H^{jk}_{\phi\phi}\right]\right)\right) \\ & +\left[G^{jk}_{\phi}\right]^{T}\left(\left[G^{jk}_{\phi}\right]^{T}\circ\left[P^{jk}\right]\right)\left[G^{jk}_{\phi}\right]\Bigg\}\end{aligned} \tag{1.14}$$

Equation (1.12) represents all of the torques resulting from system inertia (i.e. system movement). However, it does not include torques resulting from gravity effects or external forces. Equation (1.15) includes these effects where $\left[{}^{i}G^{cg}_{\phi}\right]$ represents the vertical G function to the center of gravity of the i^{th} link, $\mathbf{L}_{g,i}$ represents the load due to gravity, $\left[{}^{i}G^{e}_{\phi}\right]$ is the G function of the i^{th} link about the EE, and $\mathbf{L}_e$ is the EE load.

$$\tau^{total}_{\phi} = \left[I^*_{\phi\phi}\right]\ddot{\phi} + \dot{\phi}^{T}\left[P^*_{\phi\phi\phi}\right]\dot{\phi} + \sum_{i=1}^{n}\left[{}^{i}G^{cg}_{\phi}\right]_{n}\mathbf{L}_{g,i} + \left[{}^{i}G^{e}_{\phi}\right]\mathbf{L}_{e} \tag{1.15}$$

1.2. MOTION PLANNING

Motion Planning involves the generation of time-based trajectories for robotic manipulators. This can be done in joint space or Cartesian space. In joint space, the motion of each joint axis is programmed independently. This is sufficient for moving between two positions when the end-effector path between the two positions is not important. However, Cartesian space trajectories are necessary for performing more complex tasks and are the focus of this research. The next section provides a brief

description of joint space planning. Then, a review of some common Cartesian space planning methods is presented.

1.2.1. Joint Space Planning

In joint space planning, each actuator is treated like a one degree of freedom (DOF) system, and their trajectories are computed individually. Thus, in a 10-DOF robot, ten trajectories will be computed. There are many different methods for computing a smooth trajectory for 1-DOF. Most of these methods involve a smooth ramp-up to a maximum velocity at the beginning of the trajectory, and a smooth ramp-down to zero velocity at the end. Some of the popular trajectory generation methods include: polynomial, sinusoidal, and trapezoidal. A more thorough examination of them can be found in [38].

Joint space planning has many advantages. First, joint space planning is simple mathematically and can be calculated quickly. Second, since robotic systems take joint angles as inputs, it is convenient to generate trajectories with respect to joint angles. Also, harm to individual actuators is reduced, because a smooth motion is planned for each joint. This will allow for increased performance and less repairs. However, joint space planning does not allow for control over the EE during its motions. This makes it unsuitable for use in generating trajectories for complex tasks.

1.2.2. Cartesian Space Planning

Cartesian space planning involves developing the position and orientation trajectories for the manipulator end-effector. The description of the end-effector in space has six degrees of freedom (DOF) and each of these must be controlled. These end-effector positions are then converted into joint positions at discrete instances in time and sent to the robot controller. In the past, there have been numerous different approaches to Cartesian space planning. This section will highlight some of the major areas to demonstrate a need for a more geometric approach.

One of the main problems with Cartesian space motions is that a smooth end-effector motion may provide an undesirable motion at the joint level. One way this problem has been addressed is to approximate the Cartesian path with a smooth joint trajectory. This is done by dividing a given end-effector path into a set of *n* knot points. Then, the joint position at each knot point is determined by inverse kinematics and a curve can be fit through these positions for each joint. A variety of methods have been used for generating this curve including cubic splines [28], B-splines [52], and trigonometric splines [45]. While these methods are adequate for generating smooth joint trajectories that approximate Cartesian paths, they still produce errors and anomalies in the higher-order properties in the path that may not be tolerable for high-performance tasks.

Another popular method of Cartesian space planning is to "blend" multiple path segments together. An early method of this was introduced by Paul [34] for smoothly transitioning between multiple straight-line trajectories. Later, these techniques were generalized and formalized for more complicated motions [29][54]. The basic idea of these techniques is to transition from one specified trajectory to another using a form similar to $X(t) = \alpha(t) X_2(t) + (1 - \alpha(t)) X_1(t)$. The blending function, $\alpha(t)$, must be chosen carefully to meet initial and final constraints. For example, using the polynomial $\alpha(t) = 6t^5 - 15t^4 + 10t^3$ will satisfy initial and final velocity and acceleration constraints. Figure 1.2 shows an example of a trajectory blend. The choice of the blending function and the size of the blend will have an impact on the final shape of the trajectory. However, these techniques do not offer a great deal of control over the shape of the velocity and acceleration profiles in the blending interval.

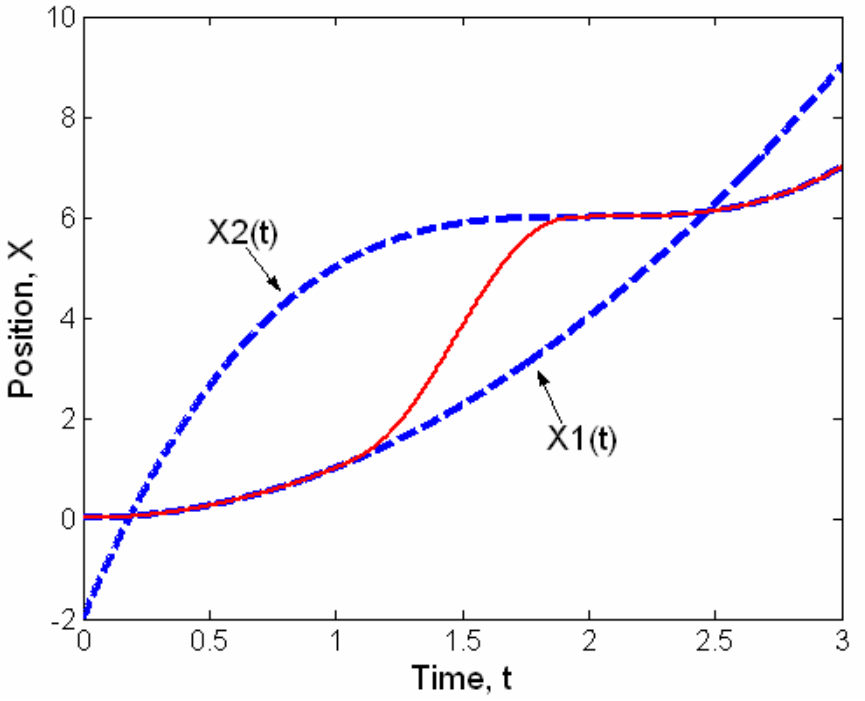


Figure 1.2. Blended Trajectory

Surface following is another useful area of research. This involves planning the trajectory for a robot manipulator based on a defined surface (e.g. a part to be machined). This surface can be provided as a parametric description or sometimes as a CAD-model. A general survey of surface tracing can be found in [40]. Two different techniques of automating a spray-painting system based on surface following can be found in [12] and [46]. In these techniques, the motion of the end-effector along the surface is designed to apply an equal coating of paint across the entire surface. These techniques are useful for specific applications, but they do not provide a way to define general motions.

A popular method for planning the orientation trajectory of the end-effector is to base the orientation on the geometry of the curve. This is done by planning rotations relative to the Frenet Frame [3][57][58]. The Frenet Frame consists of three orthogonal vectors: the tangent vector, the normal vector, and the bi-normal vector. These vectors are defined by the shape of the curve, and the frame will move along the curve continuously

as long as the curvature and torsion are continuous. One example of how these techniques could be used would be to define a constant velocity around the normal vector on the surface in a surface tracing task. These methods are very useful in coupling the rotational motion of the manipulator with the geometry of the spatial curve and will be revisited later in discussions on rotational motion planning. However, these methods assume the geometry of the path is already provided.

The differential properties of ruled surfaces have also been used to study manipulator end-effector motions. One of the first methods of doing this was presented by Ryuh and Pennock [39]. A ruled surface can be defined as shown in Equation 1.1 where u is an independent parameter defining motion along curves $\mathbf{r}(u)$ and $\mathbf{R}(u)$ and υ defines a point on the line between the two curves. In a manipulator, $\mathbf{r}(u)$ (often called the Directrix) defines the motion of the end-effector tool tip, and $\mathbf{R}(u)$ (often called the Indicatrix) defines the motion of a point along the tool axis line. Thus, the ruled surface becomes the surface traced by a vector aligned along the tool axis. The geometric properties of this surface are then used to define the time-based motion of the manipulator end-effector. More recent works in using ruled-surface properties for generating manipulator motions can be found in [59][60][61][62].

$$\mathbf{X}(u,\upsilon) = \mathbf{r}(u) + \upsilon\mathbf{R}(u) \qquad \textbf{1.1}$$

The above survey of work shows that there has been ample research in generating time-based trajectories from provided geometries. However, the literature has been less focused on actually generating the geometry of paths. These paths are generally assumed to come from some CAD-based model or some basic interpolating scheme (polynomial curves, Bezier curves, B-Splines, NURBS, etc). Thus, the focus of this research is on studying and generating the geometry of spatial curves with emphasis on local properties.

1.3. RESEARCH OBJECTIVES

This work will concentrate on studying and generating trajectories based on the intrinsic physical properties of spatial curves: curvature and torsion. This differs from the traditional spline-based methods in that those methods tend to be defined in terms of some independent parameter with little physical meaning. The main hypothesis of this research is that focusing on the higher-order properties of spatial curves will provide a more intuitive method of generating complex motions**.** This section of the report will go into more detail on the specific research plans for this work. As mentioned earlier, the end goal of this research is the development of an intuitive method for generating complex spatial paths. The main motivation for this work is to provide an alternative way of looking at spatial trajectory planning by generating spatial curves based on the higher-order properties of curvature and torsion whose values correlate closely with the physical nature of a family of curves based on those values.

1.3.1. Motivation

This research draws motivation from the techniques used for developing coupler curves for mechanisms as these techniques also emphasized the geometry of a path in the planning phase. The desired motion curve for a planar mechanism can be specified in terms of a series of Multiply Separated Positions (MSP). A generalized algebraic solution to this problem was developed in [47][48][49]. MSPs are defined as a combination of Finitely Separated Positions (FSP) and Infinitesimally Separated Positions (ISP). Two FSPs are designated as (P-P), and two ISPs are designated as (PP). So, three Multiply Separated Positions could be defined as PPP, PP-P, or P-P-P. An FSP is simply a discrete position in the plane as shown in Figure 1.3. The position of point A in frame Σ can be found using Equation 1.2. In general, $\alpha, \beta = f(\phi)$ with ϕ being the independent

parameter. Thus, a P-P-P motion specification would simply define three discrete points on the plane for the motion curve to pass through.

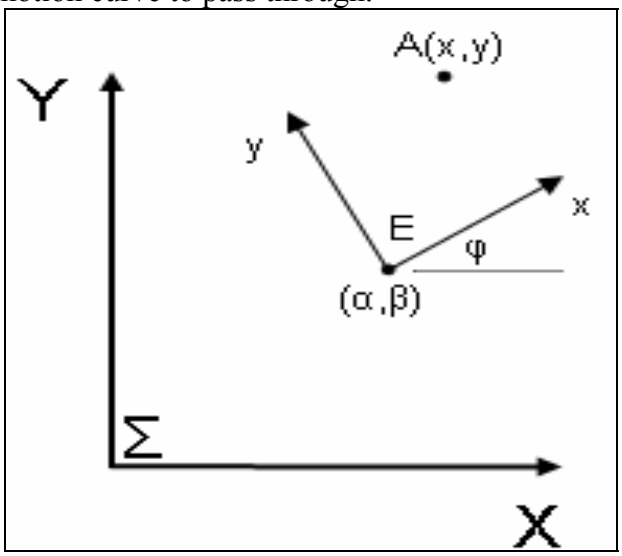


Figure 1.3. Finitely Separated Position

$$X = x\cos\phi - y\sin\phi + \alpha$$
$$Y = x\sin\phi + y\cos\phi + \beta \qquad \textbf{1.2}$$

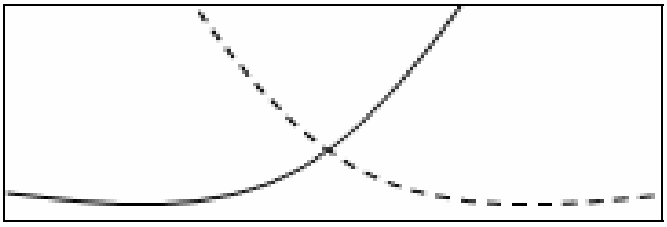


Figure 1.4. Zero Order Contact

To understand the concept of ISPs, the idea of order of contact should be defined. Two curves are said to have zero-order contact if they intersect at a single point as shown in Figure 1.4. In this case, the curves share a position but no higher-order properties.

In first order contact, two curves share two infinitesimally separated points. Consider the two curves shown in Figure 1.5 with two intersections. As the distance between these two intersections approaches zero, the two curves share a common point and a common tangent. Thus, specifying a PP MSP is equivalent to specifiying a position and a tangent for the curve.

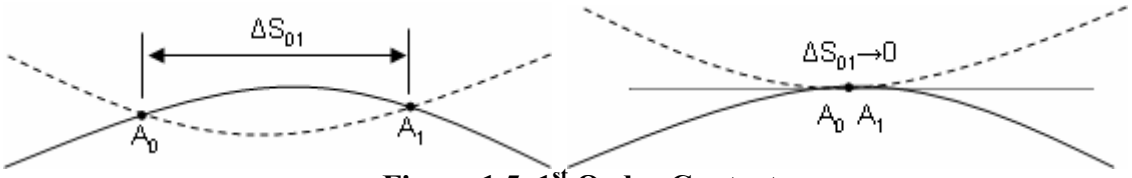


Figure 1.5. 1st Order Contact

For second order contact, two curves share three infinitesimally separated points. Consider the two curves shown in Figure 1.6. As the distances ΔS_{01} and ΔS_{12} approach zero, the two curves share a common point, tangent, and curvature. It can be similarly shown that 3rd order contact involves an additional shared first derivative of curvature (κ').

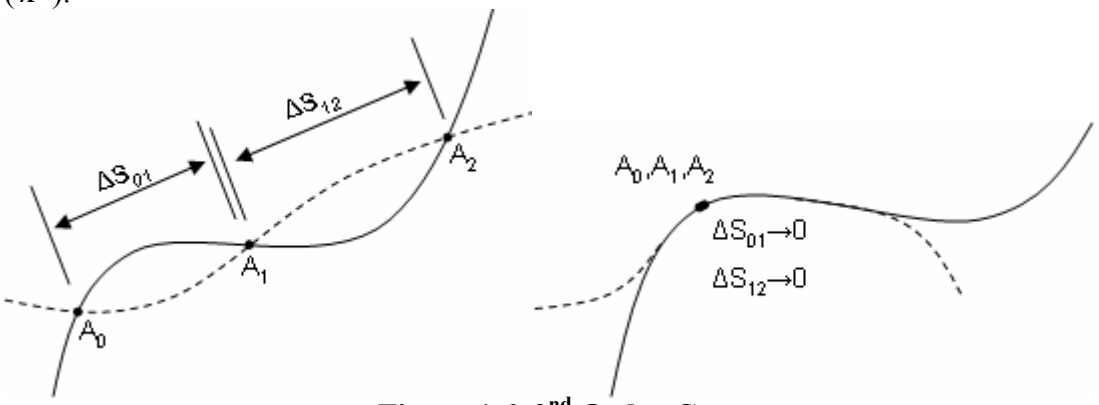


Figure 1.6. 2nd Order Contact

Once the motion constraints have been specified, the coupler curve and mechanism constraints can be determined algebraically. It should be noted that the number of MSPs that can be specified is limited by the mechanism. However, a robot can be reprogrammed to follow any path that is within its workspace, and the design of the path can be decoupled from the physical system in terms of generating the shape of the path. This allows more freedom in the design of the curve for manipulators. Thus, while the actual procedure developed in this research is much different from coupler curve design, the underlying motivation is the same: designing curves based on geometric constraints.

1.4. CONCLUSIONS

This chapter began by describing the basic domain of robotics and motion planning. Then, a brief review of some of the areas of research for manipulator motion planning demonstrated a lack of focus on the geometric design of curves. Thus, to pursue the goals of this research, we will begin by studying of the theory of Algebraic Curves in Chapter 2.

This study will focus on the basic descriptions, representations and properties of Algebraic Curves. Four different representations will be explored: Implicit, Parametric, Arc-Length Parameterization, and Curvature/Torsion profiles. For each representation, the formulations for the intrinsic properties of curvature and torsion will be investigated. Then, the various advantages and disadvantages of these representations will be summarized. This chapter should supply the reader with the necessary mathematic background to understand the formulations developed in later chapters.

In Chapter Three, various curve generation techniques from other disciplines (e.g. Computer-Aided Design and Computer Graphics) are explored. These include fairly simple methods such as Bezier Curves and B-Splines as well more involved methods such as Beta-Splines and A-Splines. While all of these methods provide adequate ways to produce visually pleasing curves, the goal of this research is on developing more physically-based curve constraints.

This goal is pursued in Chapter 4 by further studying the intrinsic properties of curvature and torsion and their affects on the local geometry of curves. This starts with a study of very simple planes shapes (parabolas, circles, ellipses, etc). For each shape, variable parameters are identified that can be used to generate families of curves. Then, closed-form solutions for curvature in terms of these parameters are formulated to better develop the relationships between this property and simple planar shapes. Then, a similar

analysis is pursued for more complex spatial shapes. In this case, it becomes increasingly difficult to directly relate parametric/implicit forms of curves with the geometric properties. Thus, to study these properties, a method to directly generate curves based on curvature/torsion values is presented. This method is used to develop local surfaces defined by families of curves to illustrate the local affect of curvature and torsion on the geometry of spatial curves.

Once a physical understanding of curvature and torsion has been developed, Chapter 5 then shows how to convert these geometric constraints into parametric constraints that can be used to generate spatial curves. This involves a step-by-step formulation for developing these constraints up to the fourth order, and specific examples are included at each step that demonstrate how this process can be used. The end result is a method wherein a user/operator can provide geometric constraints (i.e. constraints based on curvature and torsion) at a set of frames, and the resulting parametric constraints can be calculated. An introduction to several potential methods for blending between these constraints (polynomial, trapezoidal, etc) is also introduced and elaborated in Appendix B.

Chapter Six then takes this method for developing the geometric shape of a curve and implements it inside of a manipulator controller. First, the basic software framework for a Motion Planner previously designed at the RRG will be presented. Then, specific integration issues, such as defining the motion along the curve and rotational planning methods, are discussed. This provides a useful, robot independent testbed that can easily be used and expanded in future work. Finally, Chapter Seven will present a summary of this work as well as suggestions for future work in this area. This will also demonstrate how the techniques developed in this work may be applied to specific manipulator tasks.

2. CHAPTER TWO

Algebraic Curves

This chapter will introduce some of the necessary mathematical background for this research, which involves looking at the various different representations for algebraic curves and their properties. In the first section, implicit forms for algebraic curves will be discussed (i.e. $f(x,y)=0$). These curves are basically defined to be the set of all points (*x,y*) that are a solution to the given algebraic equation. Next, the standard parametric form for algebraic curves will be discussed. In this form, the *x*, *y*, and *z* coordinates are defined as functions of some independent parameter ($x=f(u), y=f(u), z=f(u)$). For defining motion, this independent parameter *u* is often taken to be time *t* or some function of time. A special case of this form where the independent parameter is arc length will also be studied. Finally, a method of defining a spatial curve in terms of its curvature and torsion profiles will be developed. The advantages and disadvantages of these representations will then be compared with an emphasis on application to interactive path planning for physical systems (i.e. end-effector motion for programmable robot manipulators). The mathematics introduced in this chapter will be analyzed in more depth in later chapters.

2.1. IMPLICIT FORMS OF PLANAR ALGEBRAIC CURVES

An implicit planar curve is defined as the zero of a bivariate function $f(x,y)=0$. An *algebraic* curve is simply the case where the function $f(x,y)$ is a polynomial in *x* and *y* with scaling coefficients a_{ij}. A degree *n* algebraic curve is thus defined as shown in Equation 2.1 where $n=i+j$. Then, the curve represents the set of all points (*x,y*) that are

solutions to this equation. For example, Figure 2.1 shows a curve represented by the equation $y\left(x^2+1\right)-1=0$.

$$f\left(x,y\right)=\sum_{i+j}^{n}a_{ij}x^{i}y^{j}=0 \qquad \textbf{2.1}$$

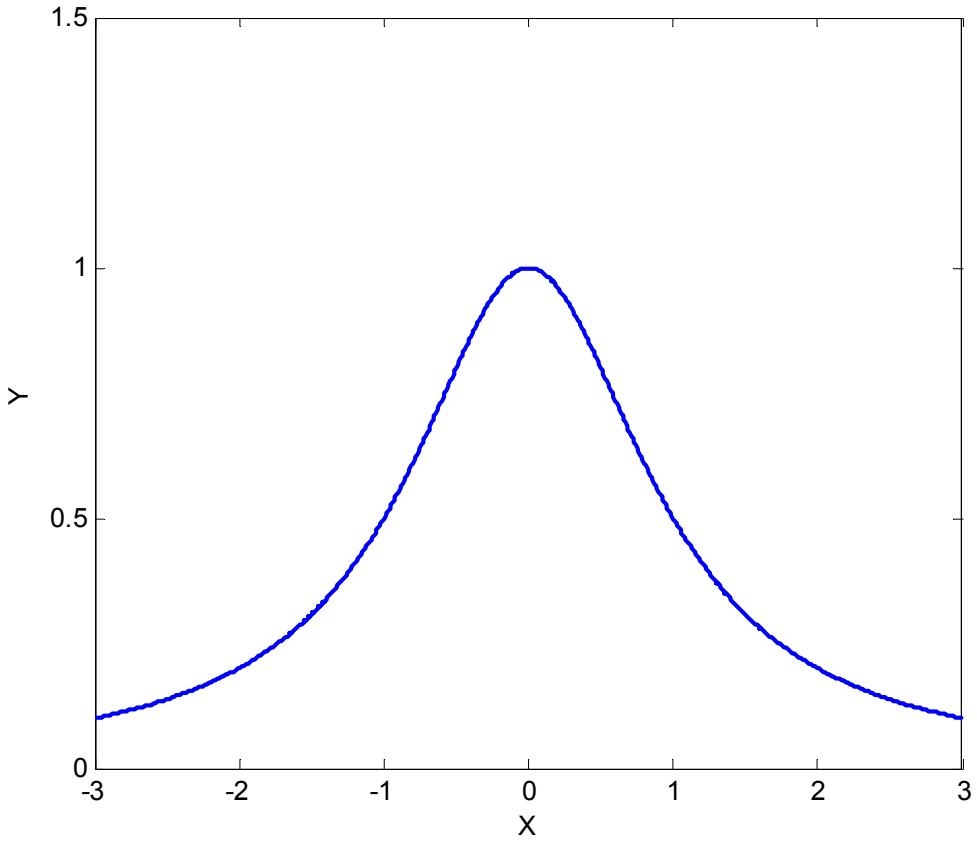


Figure 2.1. Implicit Algebraic Curve

2.1.1. Basic Properties of Implicit Curves

This section will discuss two simple, but important, properties of implicit planar curves: curvature and inflection points. Curvature is the reciprocal of the local radius of curvature. Thus, it is basically a measure of how much the curve is "bending". A curvature of zero would mean that the curve is a locally a straight line, and a high curvature would indicate a sharp bend in the curve. In the limit, when the curvature is infinite, the curve becomes a cusp. The equation for curvature of an implicit planar curve

is shown in Equation 2.2 where $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$, $f_{xx} = \frac{\partial^2 f}{\partial x^2}$, $f_{yy} = \frac{\partial^2 f}{\partial y^2}$, and $f_{xy} = \frac{\partial^2 f}{\partial x \partial y}$.

$$\kappa(x,y) = \frac{f_{xx} f_y^2 - 2 f_{xy} f_x f_y + f_{yy} f_x^2}{\left(f_x^2 + f_y^2\right)^{3/2}} \qquad \textbf{2.2}$$

An inflection point is closely related to curvature and occurs whenever the sign of curvature changes. This represents the curve changing direction. Another way to think of curvature is to examine how the tangent of the curve is changing as the curve is traced. The tangent will rotate in the counter-clockwise direction if the curvature is positive and in the clockwise direction if the curvature is negative. Thus, the tangent vector will "cut" the curve at an inflection point [16]. For example, in Figure 2.2, the curve on the right contains an inflection point at the origin while the curve on the left does not.

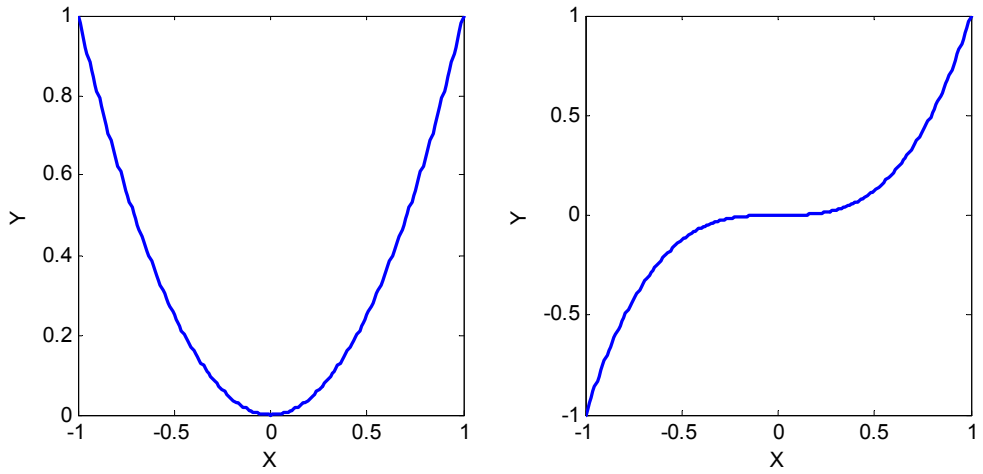


Figure 2.2. Example of an Inflection Point

2.1.2. Genus and Singular Points

An important categorization of an algebraic curve is its *genus*. The genus of an algebraic curve of degree *n* is given by Equation 2.3 and will always be greater than or equal to zero. Thus, a quadratic curve (n=2) will have no singular points, and a cubic

(n=3) will have at most one. An important result relating to genus is that an algebraic curve will have a rational parameterization if and only if the genus is equal to zero [55]. This is an important result as a parameterization allows for easier definition of motion along a curve. A basic procedure for obtaining this parameterization will be presented in the following section on parametric curves.

$$genus = \frac{(n-2)(n-1)}{2} - \sum \text{singularities} \qquad \textbf{2.3}$$

It not becomes necessary to expand on the meaning of singularities on algebraic curves. A singularity on an algebraic curve is defined to be any point on the curve where the both first derivatives vanish ($\frac{\partial f(x,y)}{\partial x} = \frac{\partial f(x,y)}{\partial y} = 0$). Note that these points must still lie on the curve. For example, the curve $x^3 + y^3 - 1 = 0$ does not contain any singularities even though $\frac{\partial f(0,0)}{\partial x} = \frac{\partial f(0,0)}{\partial y} = 0$, because the point x=0, y=0 does not lie on the curve.

Now, to study a few of the most common types of singularities, we will consider the algebraic curve given by the equation $y^2 = ax^2 + x^3$. It is easy to see that for all values of a, there will be a singular point at the origin. We will consider three cases (a<0, a=0, a>0) that will lead to the three different types of singularities.

2.1.2.1. Double Point

In the first case, we choose a to be a positive value. This leads to a double point. At a double point, the curve crosses itself but does not share tangents between the crossing branches. An example of a double point is shown in Figure 2.3.

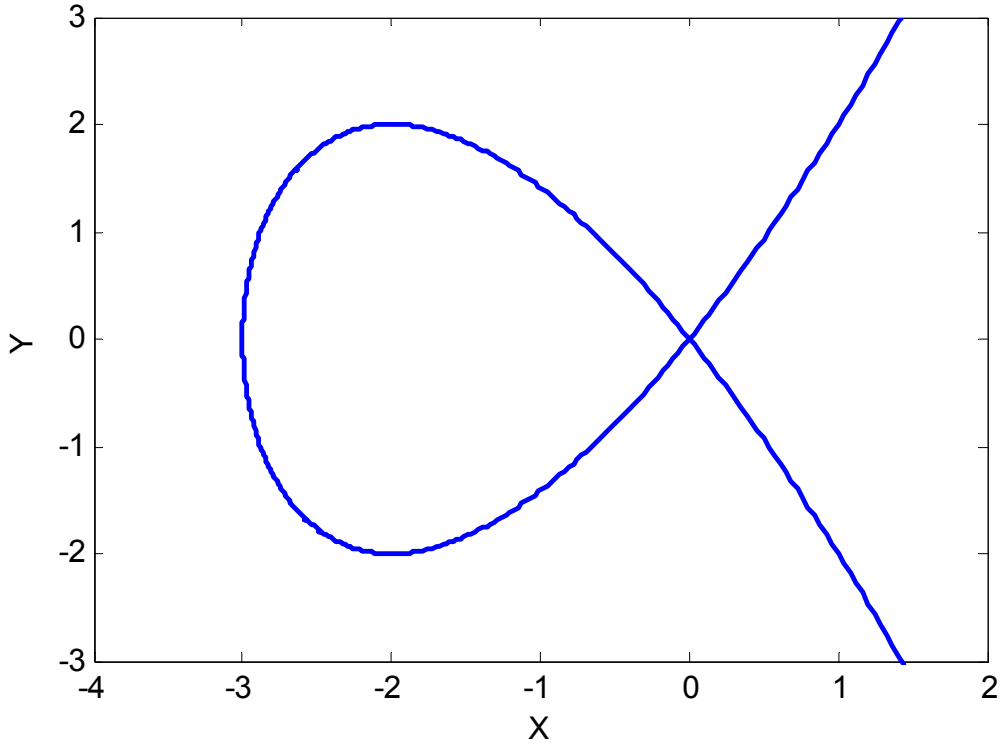


Figure 2.3. A Double Point in the curve: $y^2 = x^3 + 3x^2$

Another condition for a double point is that the determinant of the Hessian matrix of second derivatives will be negative definite at the singular point. For a curve in two variables, the Hessian matrix is shown in Equation 2.4. The calculation of the determinant of this matrix can be found be plugging in the partial derivative values as shown in Equation 2.5. Plugging in the singular point $(x, y) = (0, 0)$, the determinant of this matrix for this curve at the singular point is -4*a*, which will be negative for all positive values of *a*.

$$\begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} \qquad \textbf{2.4}$$

$$\begin{vmatrix} -2a-6x & 0 \\ 0 & 2 \end{vmatrix} = -4a-12x \qquad \textbf{2.5}$$

2.1.2.2. Cusp

A cusp occurs in this curve when a is set to zero. At a cusp, two branches of the curve meet with a shared tangent, and the determinant of the Hessian matrix is zero. The deriviative of the Hessian will be the same as before (-4a) but with $a=0$ this time. Figure 2.4 shows an example of a cusp.

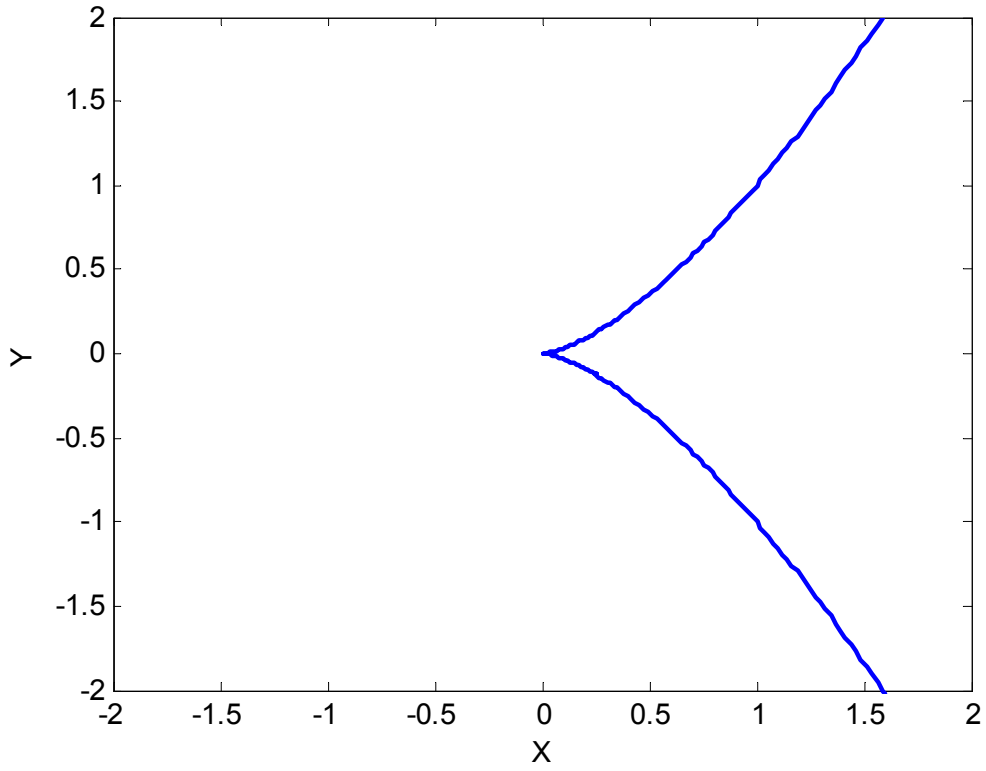


Figure 2.4. A cusp in the curve: $y^2 = x^3$

2.1.2.3. Isolated Point

Another kind of singularity is known as an isolated point. This occurs when the equation of the curve has a point that is disconnected from the rest of the curve. Consider the case when a=-3 shown in Figure 2.5.

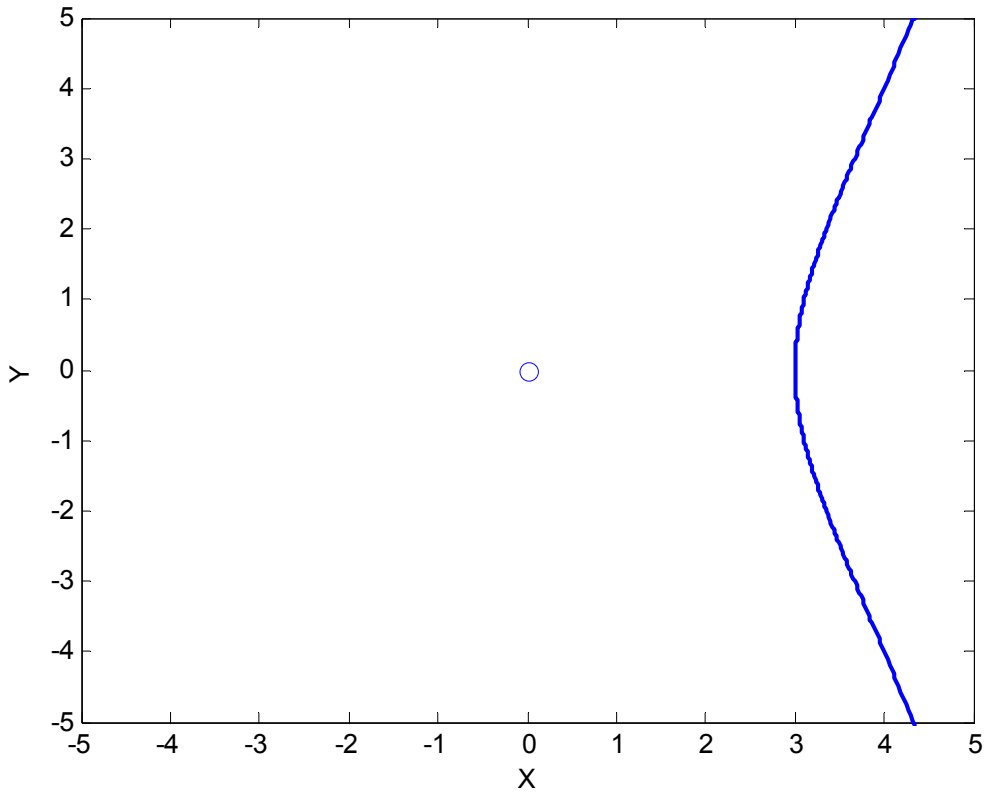


Figure 2.5. Isolated Point on curve: $y^2 = x^3 - 3x^2$

As seen in this plot, most of the curve lies to the right of the origin. However, it is clear the point x=0, y=0 is also a solution to the algebraic equation. This leads to an isolated point. At an isolated point, the determinant of the Hessian matrix will be positive definite. Once again, the determinant of the Hessian will be -4a. However, since a is now negative, this will always be a positive value.

2.1.2.4. Other Singularities

For higher degree curves, singular points are often a form of one of the types of singularities previously discussed but with a higher multiplicity. For example, the degree four curve shown in Figure 2.6 has a triple point at its origin. It should also be noted that

not all singular points have an obvious geometric feature; it is also possible for a curve to look smooth at a singularity.

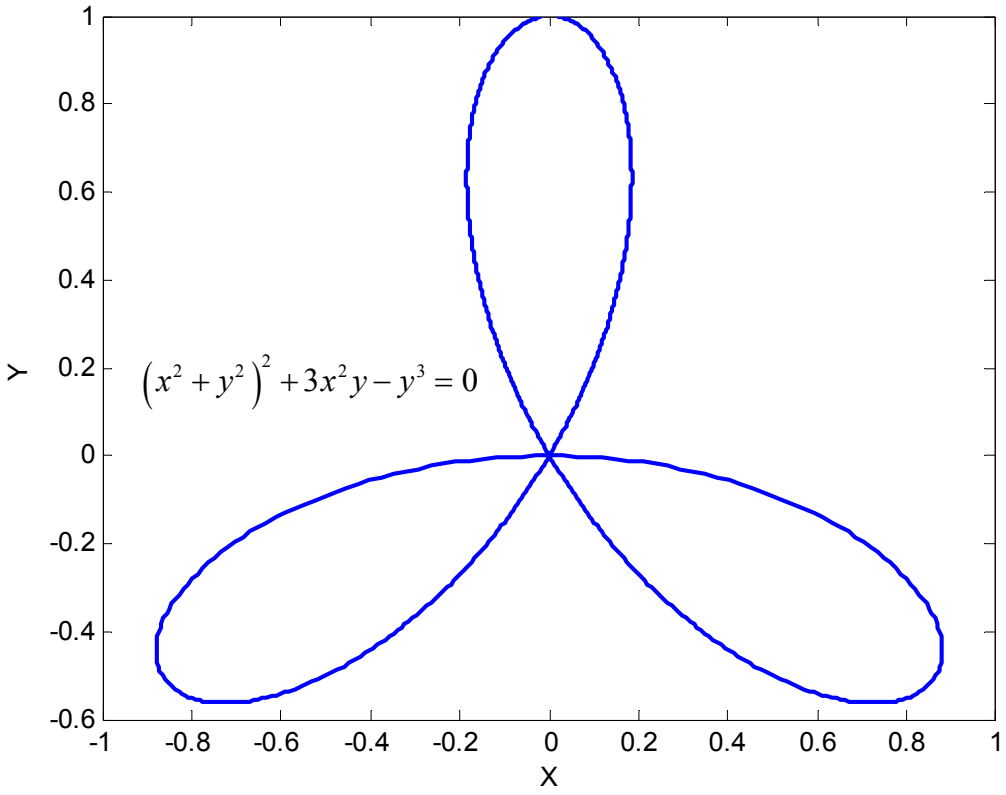


Figure 2.6. Degree Four Curve with a Triple Point

2.1.3. Implicit Spatial Curves

This section has so far focused on planar forms of implicit curves. A spatial curve in implicit form is defined as the intersection of two implicit surfaces (i.e. $f\left(x,y,z\right)=0 \cap g\left(x,y,z\right)=0$). A simple example of this is shown in Equation 2.6 where a curve is defined by a unit sphere intersecting with the z=0 plane. Thus, this curve is simply a unit circle in the xy plane. Defining a curve in this manner is important for many applications, such as looking at the curves resulting from intersecting two complex surfaces. However, in general, it is difficult to define a desired curve shape (useful for motion planning) in this manner.

$$\begin{aligned} x^2 + y^2 + z^2 - 1 &= 0 \\ z &= 0 \end{aligned} \qquad \textbf{2.6}$$

2.1.4. Summary

Algebraic curves defined in implicit forms can provide a good mathematical understanding of the curve and have a wealth of historical literature and research associated with them. However, they have several disadvantages in terms of curve generation as it can be difficult to describe a motion along the curves in terms of their point parameters. This problem becomes even more difficult when trying to describe spatial motions. For this reason, implicit curves are often converted to parametric form. The next section will describe a basic procedure for doing this for planar curves and then go into more depth on the properties of parametric curves.

2.2. STANDARD PARAMETRIC CURVES

A planar parametric curve involves defining the *x* and *y* coordinates with respect to some independent parameter over a certain range as shown in Equation 2.7. Curves defined in this manner are in general easier to work with than implicit curves. Thus, implicit curves are often converted into a parametric form for the purposes of rendering or defining a motion along the curve. The basic procedure for this conversion is introduced for low degree curves in the following section.

$$\left.\begin{aligned} x &= f(u) \\ y &= f(u) \end{aligned}\right\}, u \in [a,b]^{1} \qquad \textbf{2.7}$$

2.2.1. Implicit to Parametric Conversion

As mentioned in the Section 2.1.2, a curve must have a genus of 0 for a rational parameterization to exist. Thus, a rational parameterization will always exist for a

[1] This notation means the independent parameter *u* is defined on some interval *a* to *b*.

quadratic curve, and a cubic curve will require the curve to have one singular point. For these simple curves of low degree, a parameterization can be found by computing the intersection of the curve with a "family of lines" passing through a point on the curve. For example, consider the simple quadratic curve which passes through the origin given by $y - x^2 - yx = 0$. Now, take the family of lines passing through the origin $y=tx$ and substitute it into the equation. The resulting parameterization is shown in Equation 2.8.

$$\begin{aligned} & y - x^2 - yx = 0 \\ & tx - x^2 - tx^2 = 0 \\ & x\left(t - x - tx\right) = 0 \\ & t - x\left(1 + t\right) = 0 \Rightarrow x = \frac{t}{1+t} \\ & y = tx \Rightarrow y = \frac{t^2}{1+t} \end{aligned} \qquad \textbf{2.8}$$

Now, consider the cubic curve with a double point (singularity) that was introduced in Section 2.1.2.1. The parameterization of this curve is shown in Equation 2.9.

$$\begin{aligned} & y^2 - x^3 - 3x^2 = 0 \\ & y = tx \Rightarrow t^2x^2 - x^3 - 3x^2 = 0 \\ & x^2\left(t^2 - x - 3\right) = 0 \Rightarrow t^2 - x - 3 = 0 \Rightarrow x = t^2 - 3 \\ & y = tx \Rightarrow y = t^3 - 3t \end{aligned} \qquad \textbf{2.9}$$

This curve was also generated by taking the family of lines passing through the singularity at the origin. From Figure 2.3, it is clear that these lines *must* pass through the origin as any other point on the curve will lead to multiple intersection points. Thus, the singularity actually allows the rational parameterization to be defined. Figure 2.7 shows two more examples of implicit versus parametric curves.

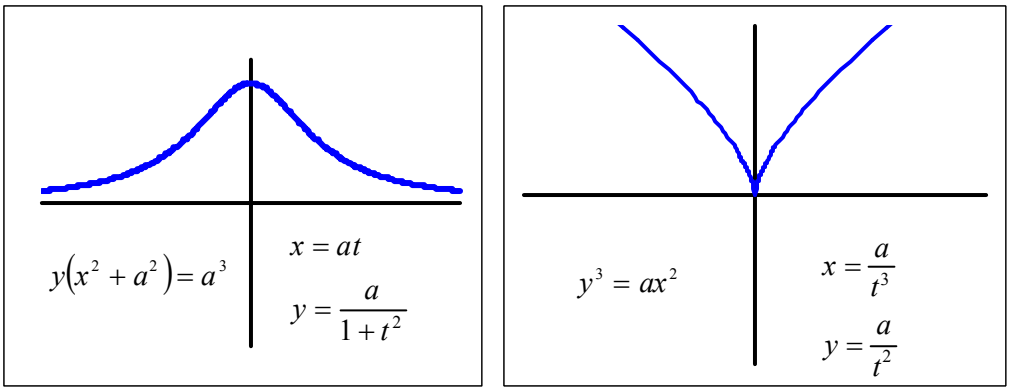


Figure 2.7. Two Examples of Implicit vs. Parameteric Representations

As the degree of an implicit curve gets higher, it becomes more difficult both to find curves of genus 0 and to find the rational parameterization of these curves. For example, a quartic curve (n=4) will require 3 singular points, and a family of curves instead of lines will often be needed to find the parameterizations. A generalized method of finding parametric description of algebraic plane curves is presented in [1]. This is further generalized to spatial curves defined as the intersection of two implicit surfaces [2]. Higher degree curves are also often approximated with piecewise segments [5]. While this research will mainly focus on curves in parametric form, it is important to recognize that methods of converting between these representations exist. Thus, implicit forms can be used when needed.

2.2.2. Spatial Parametric Curves

Parametric curves can easily be extended from planar forms to spatial forms as shown in Equation 2.10. In this form, each point on the curve can be uniquely defined by its position vector $\mathbf{r} = [x, y, z]$ measured from the origin. Likewise, a curve can be defined as a real vector function $\mathbf{r} = \mathbf{r}(u)$ where each component of $\mathbf{r}$ is also a function of the independent parameter u for some parameter range. This provides a mapping from $R \rightarrow R^3$. For generating motion along paths, this independent parameter is often taken to

be time *t*. Figure 2.8 shows two examples of spatial parametric curves along with their equations.

$$\left.\begin{aligned} x &= f(u) \\ y &= f(u) \\ z &= f(u) \end{aligned}\right\}, u \in [a,b] \qquad \textbf{2.10}$$

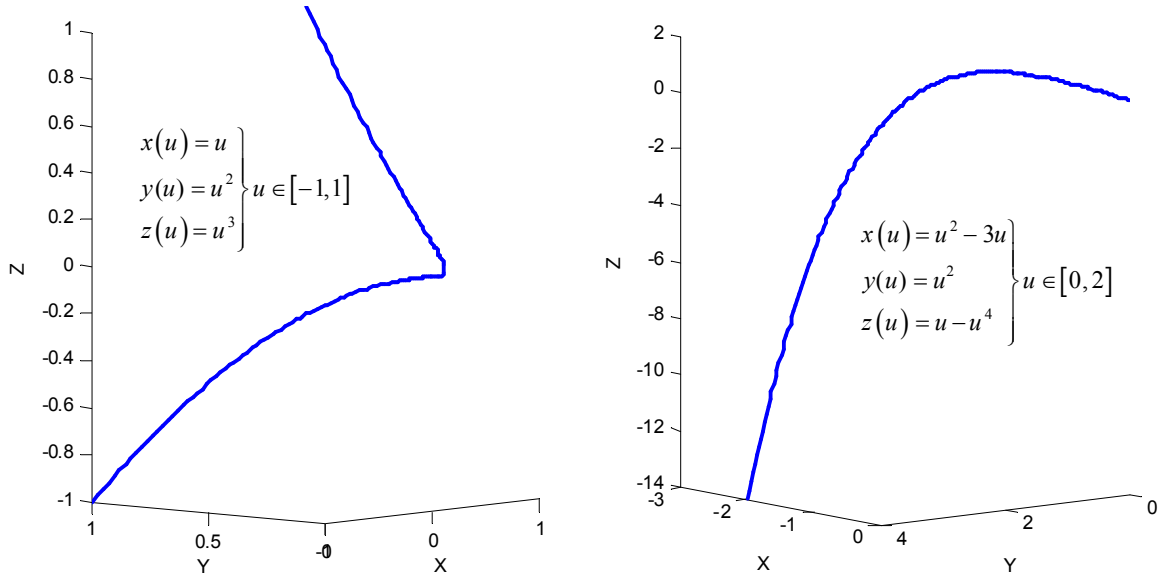


Figure 2.8. Example Spatial Parametric Curves

2.2.3. Parametric Curve Properties

This section will introduce some basic physical properties of spatial parametric curves: curvature, torsion, and the Frenet Frame. The basic physical meanings behind these properties will be introduced here and described in more detail in later sections.

2.2.3.1. Curvature

In a spatial parametric curve, curvature can be calculated as shown in Equation 2.11 where $x' = dx/du$ [26]. If the z terms in this equation were removed, the equation reduces to a similar form to the curvature provided earlier for planar parametric curves. As in a planar curve, this value is the local reciprocal of curvature and gives an indication of the bending in a curve. Thus, a zero value represents a straight line. This expression

contains derivatives up to the second order which must be defined for curvature to have meaning.

$$\kappa(u) = \frac{\sqrt{(y'z'' - y''z')^2 + (z'x'' - x'z'')^2 + (x'y'' - y'x'')^2}}{(x'^2 + y'^2 + z'^2)^{3/2}} \qquad \textbf{2.11}$$

One thing to notice in the equation of curvature for a parametric curve is that the curvature value will always be positive as opposed to being a signed value as in the planar case. This is because a spatial curve can technically bend in an infinite number of directions as opposed to just two directions in a planar curve. This can be illustrated by examining the planar curve in Figure 2.9. As the curve is traversed, the tangent vector can either move in the clockwise or counter-clockwise directions, and these two directions correspond to positive and negative curvature. However, in a spatial curve, the tangent vector can rotate in any direction. Thus, curvature is defined as the magnitude of the bending without a directions, and an inflection point in a spatial curve can not be described as a point where the sign of curvature changes.

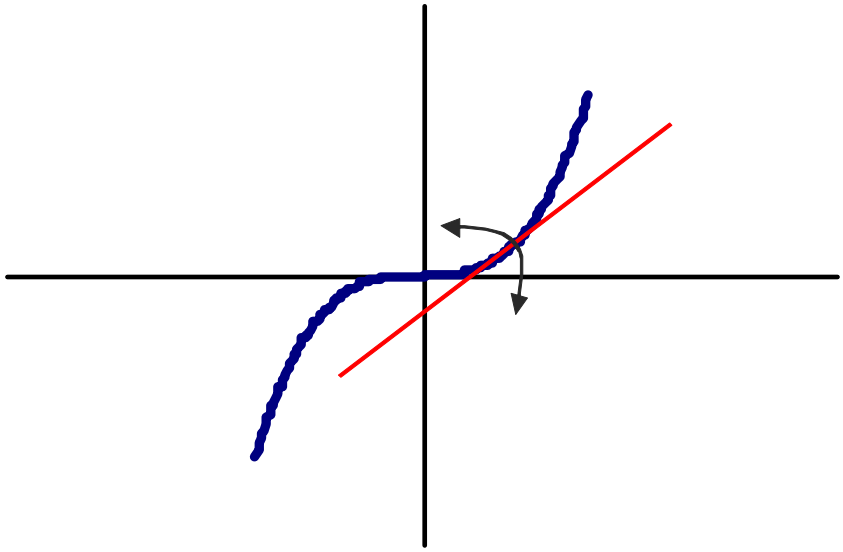


Figure 2.9. Tangent Vector Changing as Curve is Traversed.

2.2.3.2. Torsion

Torsion τ is another property of curves in space. It measures the tendency of a curve to twist out of the plane (a planar curve will have a zero torsion). Because the torsion of a planar curve is zero, τ is a strictly a property of spatial curves. The calculation for torsion is shown in Equation 2.12 for a curve in parametric form [26]. Unlike curvature, torsion has a signed value for spatial curves.

$$\tau(u) = \frac{\left(y'z''x''' - y''z'x'''\right) + \left(z'x''y''' - x'z''y'''\right) + \left(x'y''z''' - y'x''z'''\right)}{\left(y'z'' - y''z'\right)^2 + \left(z'x'' - x'z''\right)^2 + \left(x'y'' - y'x''\right)^2} \qquad \textbf{2.12}$$

2.2.3.3. Frenet Frame

The Frenet Frame is a coordinate frame attached to the curve that helps describe the geometry of the curve. It consists of three orthogonal unit vectors: the tangent, normal, and bi-normal. The equation for the unit tangent vector is shown in Equation 2.13 [26]. This vector points along the tangent of a curve and represents the "heading" of the curve.

$$\hat{\mathbf{T}}(u) = \frac{\begin{bmatrix} x' & y' & z' \end{bmatrix}}{\sqrt{x'^2 + y'^2 + z'^2}} \qquad \textbf{2.13}$$

The unit normal vector is orthogonal to the tangent vector and tends to point in the direction of the bending of the curve. Thus, when the curvature of a curve is positive, the curve will tend to move in the direction of the normal vector. The equation for the unit normal is shown in Equation 2.14 [26]. The final vector in the Frenet Frame, the bi-normal vector, can be calculated as the cross product of the tangent and normal as shown in Equation 2.15 [26]. It will be shown later that local motion in the bi-normal direction is related to a curve's torsion. Figure 2.10 shows the Frenet Frame moving along a spatial curve.

$$\hat{\mathbf{N}}(u) = \frac{\frac{d\hat{\mathbf{T}}}{du}}{\left\| \frac{d\hat{\mathbf{T}}}{du} \right\|} \tag{2.14}$$

$$\hat{\mathbf{B}}(u) = \hat{\mathbf{T}}(u) \times \hat{\mathbf{N}}(u) \tag{2.15}$$

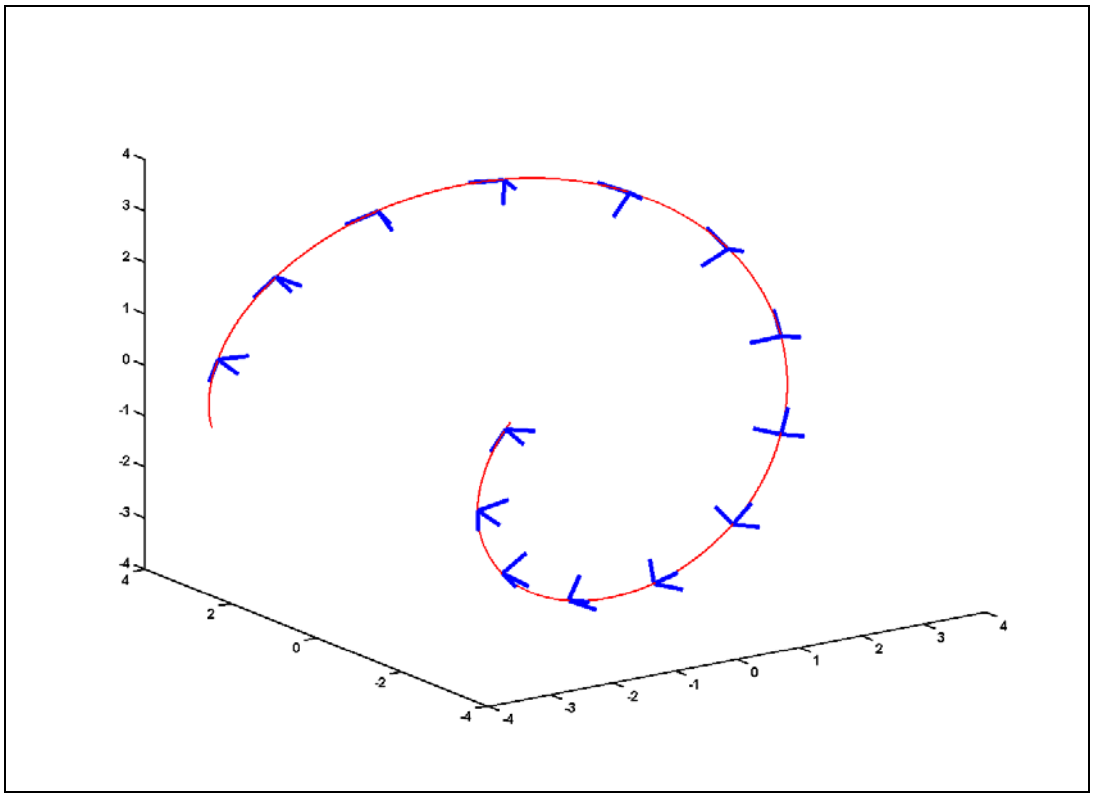


Figure 2.10. Frenet Frame on a Spatial Curve

2.2.4. Parametric Surfaces

While the main focus of this research is on the generation of curves for path planning, a brief introduction to parametric surfaces and their relationship to parametric curves is presented here. The study of algebraic surfaces is of interest in motion planning, because many robotic tasks involve interaction with surfaces (surface polishing, spray painting, etc). In parametric form, a surface can be defined as shown in Equation 2.16 where the *x*, *y* and *z* coordinates are defined as functions of two independent parameters.

$$x = f(s,t)$$
$$y = f(s,t) \qquad \textbf{2.16}$$
$$z = f(s,t)$$

In a parametric form, a surface can easily be broken down into curves that run along its surface. This can be done by setting one of the independent variables to a constant value (e.g. $\{x,y,z\} = f(s_0,t)$ or $\{x,y,z\} = f(s,t_0)$). A tangent plane can be defined at any point [*s,t*] by calculating the tangent vectors of each of these curves. The cross product of these two vectors will represent the surface normal at a given point. For example, consider the surface shown in Figure 2.11 parametrically defined as $(s,t,\cos(s+t)), s \in [-1,1], t \in [-1,1]$.

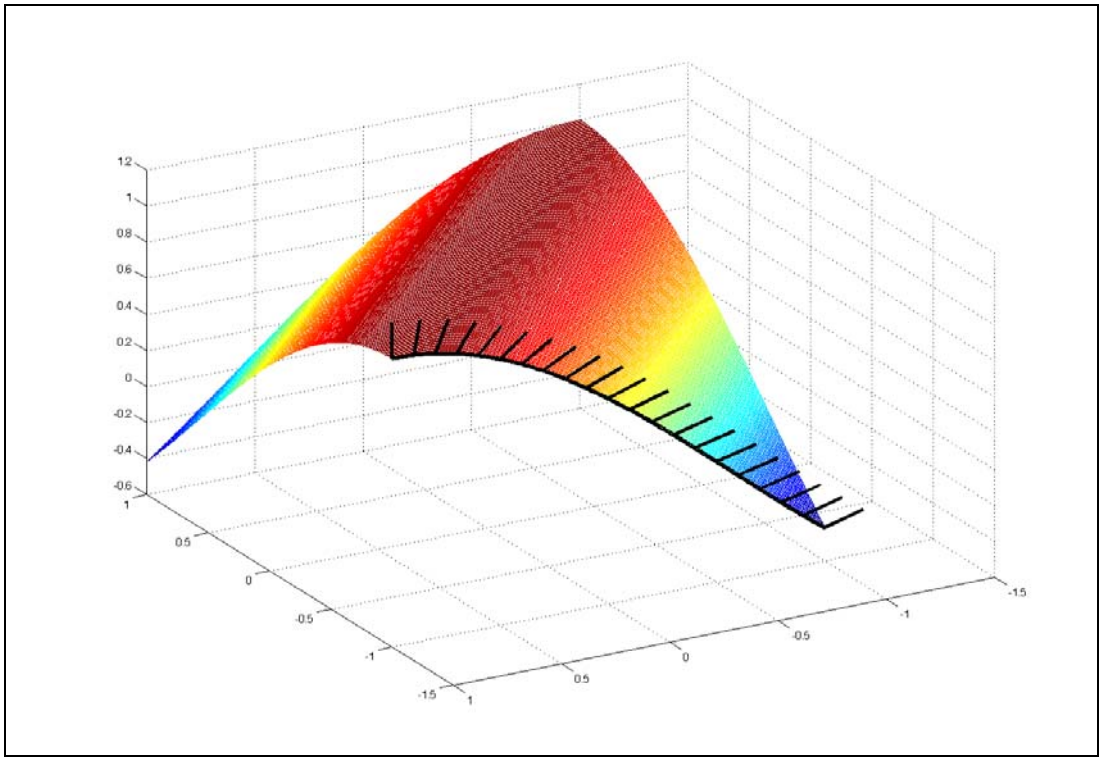


Figure 2.11. Example of a Parametric Surface

The dark line at the edge of this plot represents a parametric curve represented by $(-1,t,\cos(t-1)), t \in [-1,1]$. This curve is generated by setting *s*=-1 in the surface definition and creating a parametric equation with one independent variable. The vertical

lines on the plot represent the surface normal at any given point along this curve. This vector is calculated by taking the cross product of the two tangent vectors defined as $\mathbf{T}_1 = \left[\frac{\partial x}{\partial s}, \frac{\partial y}{\partial s}, \frac{\partial z}{\partial s}\right]$ and $\mathbf{T}_2 = \left[\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}, \frac{\partial z}{\partial t}\right]$. For this particular surface, these tangent vectors are $\mathbf{T}_1 = \left[0, 1, -\sin(s+t)\right]$ and $\mathbf{T}_2 = \left[1, 0, -\sin(s+t)\right]$. These calculations could be useful for developing motion plans for certain tasks where the end-effector orientation is a function of the geometry of the surface such as spray-painting or grinding. While this research will focus on spatial curves, it is useful to see that the same analytics apply to parametric surfaces.

2.3. ARC LENGTH PARAMETERIZATION

Spatial curves that are parameterized with respect to their arc length, *s*, are of particular interest. In these curves, the position vector and each of its components is defined to be a function of the distance traveled along the curve, $\mathbf{r} = \mathbf{r}(s)$. As well as providing a better physical meaning to the independent parameter, curves defined in this manner have many interesting properties. Equation 2.17 shows the formulation of arc length [26]. This equation simply integrates the distance along the curve to determine the arc length. Likewise, we can write the arc length as a function of *u* as shown in Equation 2.18. The arc length *s* is often called the natural parameter.

$$s = \int_a^b \sqrt{x'^2 + y'^2 + z'^2}\, du \qquad \textbf{2.17}$$

$$s(u) = \int_{u_o}^{u} \sqrt{\mathbf{r}' \cdot \mathbf{r}'}\, du \qquad \textbf{2.18}$$

In some special cases, a closed-form analytic solution for an arc length parameterization can be found. For example, consider a circular helix described by the equation $\mathbf{r}(u) = (r\cos u, r\sin u, c)$. For this curve, $\mathbf{r}' \cdot \mathbf{r}' = r^2 + c^2$ is a constant, and the integral in Equation 2.18 does not need to be computed for every value. However, most

curves cannot be easily converted into an arc length parameterization, and a numerical method is needed.

2.3.1. Arc Length Parameterization Properties

This section will show how the physical properties of spatial curves can be calculated for arc length parameterized curves. Several useful relationships can be developed from this parameterization.

2.3.1.1. Frenet Frame

As described in the previous section, the Frenet Frame is a moving coordinate frame attached to a curve consisting of three unit vectors: the tangent, the normal, and the bi-normal. The unit tangent vector can also be easily calculated from an arc length parameterization. This is shown in Equation 2.19.

$$\hat{\mathbf{T}} = \frac{\dfrac{d\mathbf{r}}{du}}{\left\|\dfrac{d\mathbf{r}}{du}\right\|} = \frac{\dfrac{d\mathbf{r}}{du}}{\sqrt{\mathbf{r}' \cdot \mathbf{r}'}} = \frac{\dfrac{d\mathbf{r}}{du}}{\dfrac{ds}{du}} = \left[\frac{dx}{ds} \;\; \frac{dy}{ds} \;\; \frac{dz}{ds}\right] \qquad \textbf{2.19}$$

Likewise, the unit normal vector can also be easily calculated as before. This is shown in Equation 2.20. The unit bi-normal can be calculated as before by taking the cross product of the unit tangent and unit normal (Equation 2.21). This provides the three vectors that make up the Frenet Frame.

$$\hat{\mathbf{N}} = \frac{d\hat{\mathbf{T}}}{ds} \Bigg/ \left|\frac{d\hat{\mathbf{T}}}{ds}\right| \qquad \textbf{2.20}$$

$$\hat{\mathbf{B}} = \hat{\mathbf{T}} \times \hat{\mathbf{N}} \qquad \textbf{2.21}$$

2.3.1.2. Curvature

The curvature κ can now be calculated from Equation 2.22 [26]. This shows that the curvature is a measure of how quickly the unit tangent is moving with respect to distance along the curve. Physically, this represents "bending" in the curve. The

reciprocal of curvature is called the radius of curvature, $\rho = \frac{1}{\kappa}$. Another interesting relationship is shown in Equation 2.23. This shows that the unit normal vector is related to the curvature and second derivative with respect to arc length. The physical meaning behind this is that the curvature is the magnitude of the change in direction of the tangent along the curve. Thus, when the curvature is zero, the tangent vector will not change and the curve will continue in a straight line. On the other hand, a high curvature will result in the tangent vector rapidly changing direction. An infinite curvature, as in the case of a cusp, thus represents a discontinuity in the tangent vector along the curve.

$$\kappa = \left| \frac{d\hat{\mathbf{T}}}{ds} \right| = \sqrt{\frac{d^2\mathbf{r}}{ds^2} \cdot \frac{d^2\mathbf{r}}{ds^2}} \qquad \textbf{2.22}$$

$$\hat{\mathbf{N}} = \frac{1}{\kappa} \frac{d^2\mathbf{r}}{ds^2} = \rho \frac{d^2\mathbf{r}}{ds^2} \qquad \textbf{2.23}$$

2.3.1.3. Torsion

Physically, torsion is a measure of the rate of change of the osculating plane relative to the governing parameter *u*. The osculating plane is defined as the plane spanned by the curve tangent and normal vectors. Thus, a constant zero torsion means that a curve will never leave the osculating plane and will be planar. This is shown in Figure 2.12.

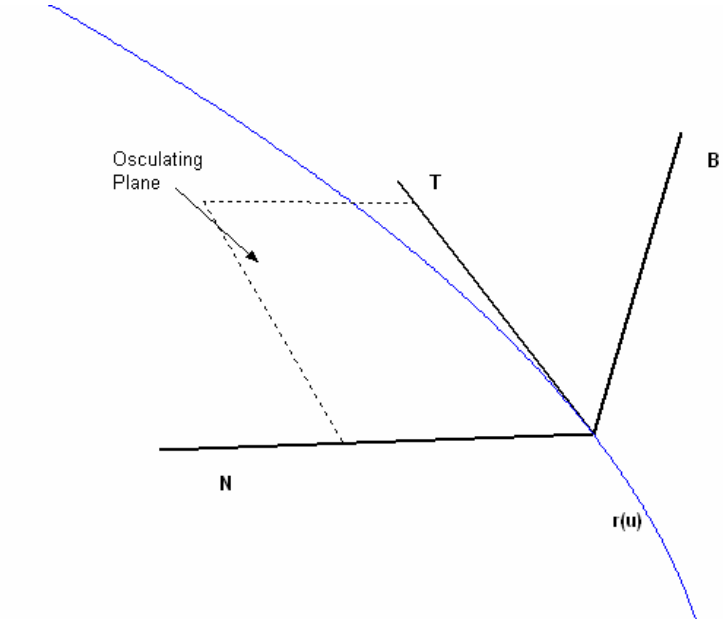


Figure 2.12. Osculating Plane of a Spatial Curve

As is shown in this plot, the bi-normal vector is perpendicular to the osculating plane. Equation 2.24 [26] shows that if the bi-normal vector is not changing, then the torsion is zero. This makes sense, because the bi-normal vector will change if the osculating plane rotates (i.e., it does not purely translate). If the bi-normal vector is not changing, this means the curve is a planar curve staying in the plane formed by the tangent and normal vectors. Torsion has a sign convention that "right-handed" curves are given positive torsion.

$$\tau = -\hat{\mathbf{N}} \cdot \frac{d\hat{\mathbf{B}}}{ds} \qquad \textbf{2.24}$$

Torsion can also be related in terms of the original curve, **r**. This is done by taking the determinant of the matrix shown 2.25. This formulation is similar to the one shown before in Section 2.2.3.2.

$$\tau = \rho^2 \begin{vmatrix} \frac{dx}{ds} & \frac{d^2x}{ds^2} & \frac{d^3x}{ds^3} \\ \frac{dy}{ds} & \frac{d^2y}{ds^2} & \frac{d^3y}{ds^3} \\ \frac{dz}{ds} & \frac{d^2z}{ds^2} & \frac{d^3z}{ds^3} \end{vmatrix} \quad \textbf{2.25}$$

2.3.1.4. Serret-Frenet Formulas

Another useful property for curves parameterized by arc length is the Serret-Frenet formulas. These formulas show the derivatives with respect to arc length of the Frenet frame as a function of the current Frenet Frame, curvature and torsion. Thus, the local geometry of a curve is fully described by its position (i.e. Frenet Frame) and its curvature and torsion. These relationships are a classic result and are shown in Equation 2.26 [26].

$$\begin{bmatrix} \frac{d\hat{\mathbf{T}}}{ds} \\ \frac{d\hat{\mathbf{N}}}{ds} \\ \frac{d\hat{\mathbf{B}}}{ds} \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}} \\ \hat{\mathbf{N}} \\ \hat{\mathbf{B}} \end{bmatrix} \quad \textbf{2.26}$$

Darboux [26] studied the rotational motion of the Frenet frame as a curve is being traversed. He derived a formula for the rotation vector of the Frenet frame moving along a curve at unit speed $(\dot{s} = 1)$ as $\mathbf{D} = \tau\hat{\mathbf{T}} + \kappa\hat{\mathbf{B}}$. This is known as the Darboux vector. In a planar curve, the first term will drop out and the Frenet frame will rotate around the binormal vector with an angular velocity $\omega = \kappa$. This formulation is useful for studying the rotational motions of a rigid body attached to the Frenet frame moving along a curve. The Serret-Frenet formulas can also be rewritten as shown in Equation 2.27.

$$\frac{d\hat{\mathbf{T}}}{ds} = \mathbf{D} \times \hat{\mathbf{T}}, \frac{d\hat{\mathbf{N}}}{ds} = \mathbf{D} \times \mathbf{N}, \frac{d\hat{\mathbf{B}}}{ds} = \mathbf{D} \times \mathbf{B} \quad \textbf{2.27}$$

2.3.2. Motion Along a Curve

Because the independent parameter in arc length parameterized curves has clear physical meaning, the motion along a curve can be easily examined. Suppose you have a curve described with respect to arc length as shown in Equation 2.28. Now, suppose that a motion program then defines the arc length as a function of time (i.e. $s = f(t)$).

$$\{x = f(s), y = f(s), z = f(s)\} \tag{2.28}$$

Let $\mathbf{P} = [x, y, z]$ be a vector describing a point along this curve. The velocity of this point, $\mathbf{v}_p$, is given by $d\mathbf{P}/dt$. This leads to the relationship shown in Equation 2.29.

$$\mathbf{v}_p = \frac{d\mathbf{P}}{ds}\frac{ds}{dt} = \frac{d\mathbf{P}}{ds}\dot{s} \tag{2.29}$$

Next, the acceleration of the point along the curve can be calculated by taking the derivative with respect to time of Equation 2.29. This is shown in Equation 2.30.

$$\mathbf{a}_p = \frac{d}{dt}(\mathbf{v}_p) = \frac{d}{dt}\left(\frac{d\mathbf{P}}{ds}\frac{ds}{dt}\right) = \frac{d\mathbf{P}}{ds}\ddot{s} + \frac{d^2\mathbf{P}}{ds^2}\dot{s}^2 \tag{2.30}$$

From algebraic curve theory, the acceleration along a curve is given by Equation 2.31. By comparing Equation 2.30 and Equation 2.31, some interesting relationships are revealed. The first is $\hat{\mathbf{T}} = \frac{d\mathbf{P}}{ds}$, and the second is $\kappa\hat{\mathbf{N}} = \frac{d^2\mathbf{P}}{ds^2}$.

$$\mathbf{a}(t) = \hat{\mathbf{T}}\ddot{s} + \kappa\hat{\mathbf{N}}\dot{s}^2 \tag{2.31}$$

The first of these relationships shows a correlation between the unit tangent vector and the first derivative $\frac{d\mathbf{P}}{ds}$ with respect to arc length s. To understand this relationship, one must recognize that the magnitude of $d\mathbf{P}/dt$ is $\dot{s}$. This leads to the description shown in Equation 2.32.

$$\hat{\mathbf{T}} = \frac{\dot{\mathbf{P}}}{|\dot{\mathbf{P}}|} = \frac{d\mathbf{P}/dt}{ds/dt} = \frac{d\mathbf{P}}{ds} \tag{2.32}$$

The second part of the equation shows a relationship among the curvature, the unit normal, and the second derivative with respect to arc length. This highlights the fact that the curvature is equal to the magnitude of the derivative of the tangent vector $\left(\kappa = \left|\frac{d\hat{\mathbf{T}}}{ds}\right| = \left|\frac{d^2\mathbf{P}}{ds^2}\right|\right)$, a result first introduced in Section 2.3.1.2.

An important area of research involves how to define motion along a curve that is parameterized with respect to an arbitrary independent parameter of the type discussed in Section 0 (e.g. $x(u), y(u), z(u)$). In these curves, the independent parameter does not have a well-defined physical meaning as in arc length parameterized curves. This means a constant speed of the parameter *u* does not lead to constant spatial speed. The process of converting these representations into suitable arc length representations is often referred to as rectification. One simple way to approximate a spatial speed along a parametric curve is shown in Equation 2.33 [63]. In this equation, $v(t)$ represents some desired velocity profile (constant, trapezoidal, etc) and Δt is the time step. Thus, at each time step, a parameter value *u* can be calculated that approximates the desired velocity. This relationship will be elaborated on in Chapter 6.

$$u_{k+1} = u_k + \frac{v(t)\Delta t}{\sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2}} \qquad \textbf{2.33}$$

2.4. CURVATURE AND TORSION PROFILES

The last curve representation that will be explored in this chapter is to define a curve by a curvature and torsion profile. This representation provides good physical meaning to the geometric shape of the curve as curvature and torsion are well understood. The following section will describe how to define and generate these curves and provide a few simple examples.

2.4.1. Formulation and Generation

As mentioned earlier, this research will study the use of curvature and torsion in designing spatial motions. The key to developing path plans based on curvature and torsion profiles are the Serret-Frenet formulas shown in Equation 2.34. These equations demonstrate that the geometric shape of a spatial curve depends entirely on its curvature and torsion profiles.

$$\begin{bmatrix} \frac{d\hat{\mathbf{T}}}{ds} \\ \frac{d\hat{\mathbf{N}}}{ds} \\ \frac{d\hat{\mathbf{B}}}{ds} \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}} \\ \hat{\mathbf{N}} \\ \hat{\mathbf{B}} \end{bmatrix} \qquad \textbf{2.34}$$

A numerical interpretation of these formulae is shown in Equation 2.35. Using these equations, a spatial curve can be generated provided an initial position and orientation (i.e. Frenet Frame).

$$\begin{aligned} \mathbf{T}_{i+1} &= \mathbf{T}_i + \kappa \mathbf{N}_i \Delta s \\ \mathbf{B}_{i+1} &= \mathbf{B}_i - \tau \mathbf{N}_i \Delta s \\ \mathbf{N}_{i+1} &= \mathbf{B}_{i+1} \times \mathbf{T}_{i+1} \\ \mathbf{P}_{i+1} &= \mathbf{P}_i + \mathbf{T}_{i+1} \Delta s \end{aligned} \qquad \textbf{2.35}$$

Thus, a Frenet Frame can be positioned and oriented as desired and the spatial curve can be generated using the curvature and torsion profiles. It should be noted that identical curvature and torsion profiles will always generate the same geometric shape, but this shape must still be positioned and oriented in space.

2.4.2. Geometric meaning

One good way to examine the effect of curvature and torsion on the local shape of a curve is to take a Taylor's expansion as $\mathbf{x}(s) = \mathbf{x}(0) + \sum_{n=1}^{3} \frac{s^n}{n!} \frac{d^n \mathbf{x}(0)}{ds^n} + \mathbf{o}(s^3)$ [26]. This function approximation is shown in Equation 2.36 where the $x1$, $x2$, and $x3$ axes correspond to the tangent, normal, and bi-normal directions respectively.

$$\begin{aligned} x_1(0) &= s \\ x_2(0) &= \frac{\kappa}{2} s^2 \\ x_3(0) &= \frac{\kappa\tau}{6} s^3 \end{aligned} \qquad \textbf{2.36}$$

These relationships show several things. First, if curvature and torsion are both zero, the curve will move in its tangent direction in a straight line. Likewise, if curvature is nonzero and torsion is zero, the curve will stay in the plane formed by the tangent and normal vectors (called the osculating plane). An important aspect of spatial curves is that curvature is always defined to be positive whereas torsion has a signed value. From the above equations, this means a spatial curve will always bend in the plane in the direction of the normal vector and will tend to leave the plane in either the positive or negative direction of the bi-normal vector depending on the torsion value.

As mentioned earlier, curvature is always defined as positive for spatial curves. This presents a problem when trying to define a spatial inflection point. In planar curves, an inflection point is a point where the sign of the curvature changes; this makes the curve begin to bend in the other direction. In a parametrically defined spatial curve, the normal vector will flip directions at an inflection point. Thus, the curve will begin to bend in a different direction but will still follow the normal vector.

2.4.3. Example

Figure 2.13 shows a simple helical curve generated using curvature and torsion profiles. In this case, the curvature and torsion value are just taken to be constant values.

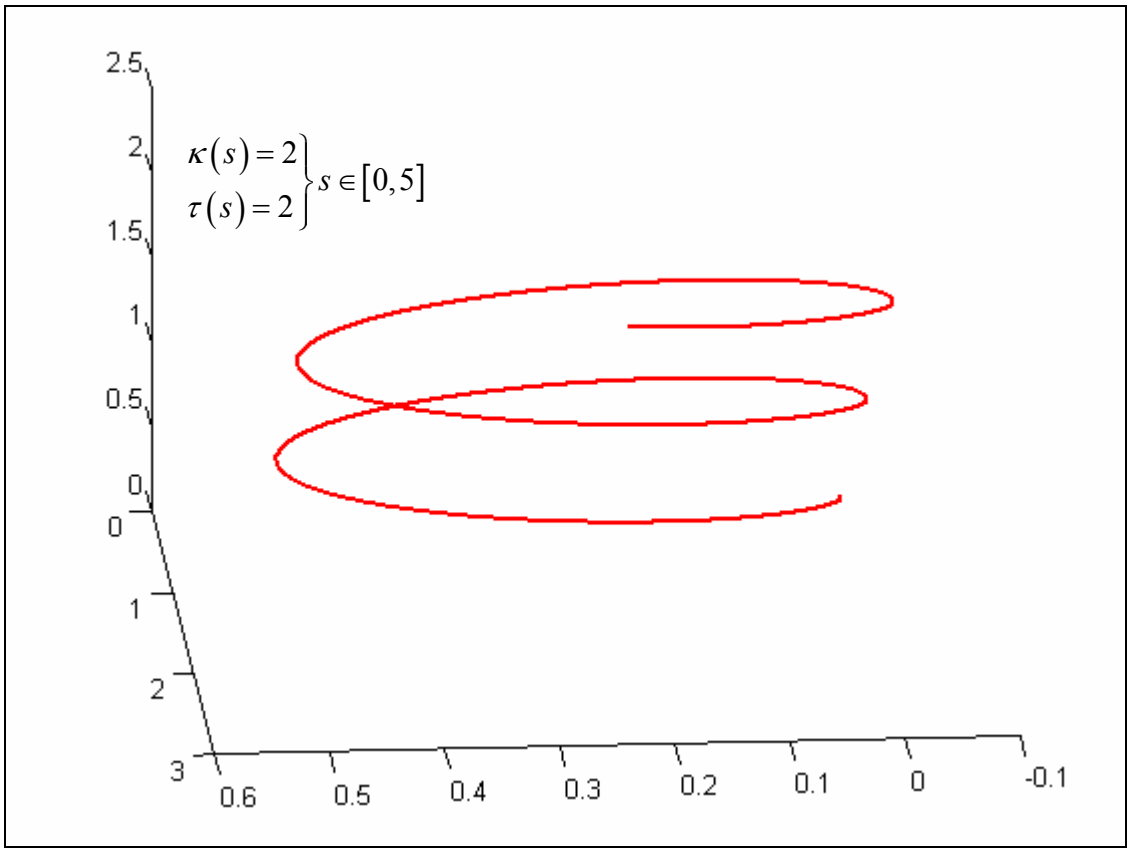


Figure 2.13. Example Helical Curve

More complicated curvature and torsion profiles can also be used to generate curves. Consider the curvature and torsion profiles shown in Figure 2.14. These plots represent the curvature and torsion values at any point along the curve as it is being traversed. Then, starting at some initial frame, the next point can be calculated using the relationships shown in 2.35 and the current value of curvature and torsion. A plot of this curve is shown in Figure 2.15.

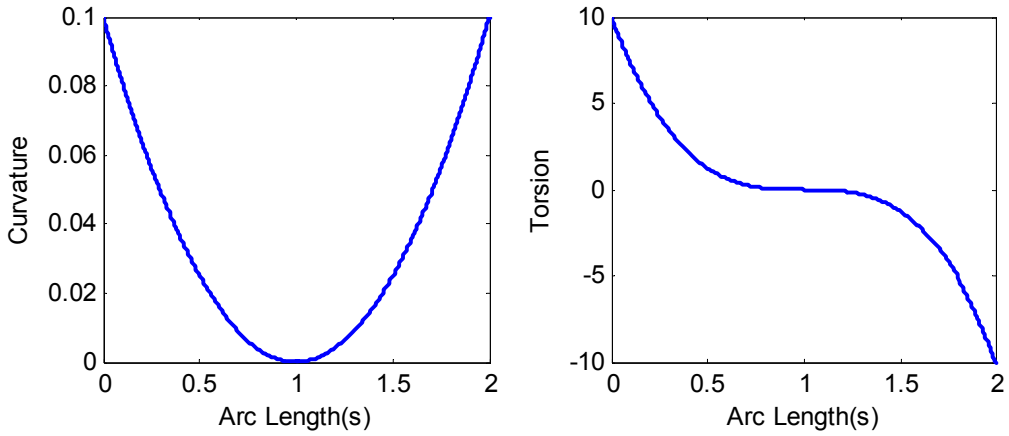


Figure 2.14. Curvature and Torsion Profiles

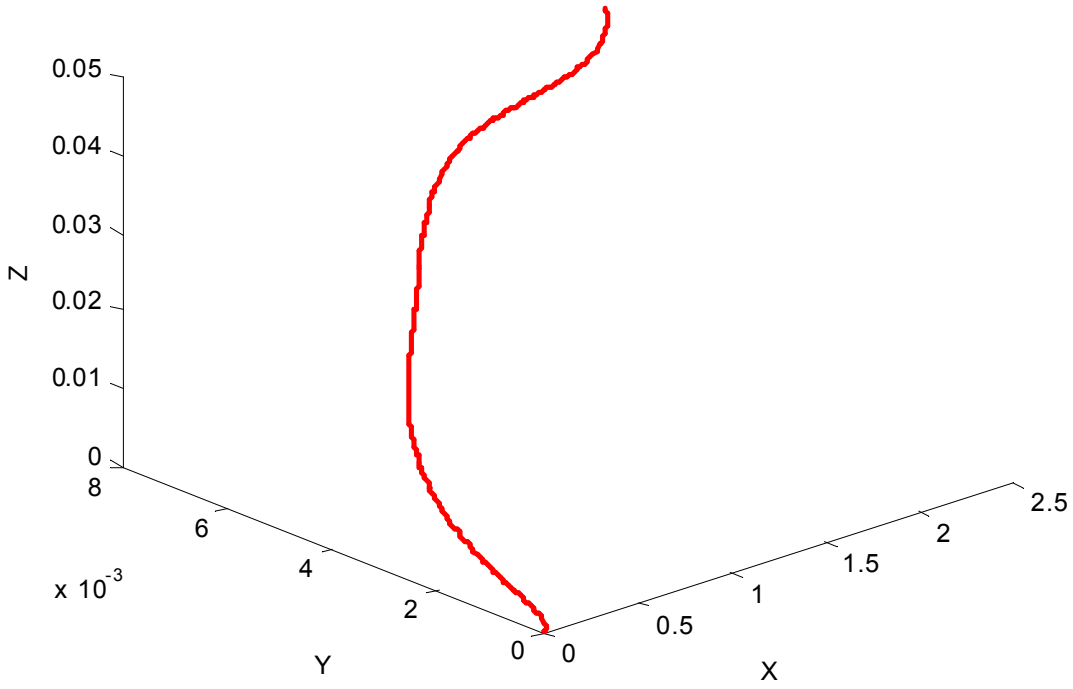


Figure 2.15. Curve Generated from Curvature and Torsion Profiles

2.4.4. Conclusions

This section described a different way of defining spatial curves by their curvature and torsion profiles. This method has several advantages. First, curvature and torsion

have clear physical meanings that could be useful in motion programming. Second, a curvature/torsion profile fully describes the geometry of a motion regardless of its position or orientation. Thus, a shape can be well defined and understood geometrically and then positioned in space as desired. However, it is difficult to meet global position-based constraints using this representation, so this form is best for defining local shapes and understanding the effects of higher-order properties.

2.5. COMPARISON OF REPRESENTATIONS

In this section, the various representations presented in this chapter will be compared based on their advantages and disadvantages. It should be noted that where possible it is always preferable to preserve as many representations for a curve or shape as possible. Table 2.1 summarizes the four representations presented in this chapter.

	Planar	**Spatial**
Implicit	$f(x,y)=0$	$f(x,y,z)=0 \cap g(x,y,z)=0$
Standard Parametric	$\left.\begin{aligned} x &= f(u) \\ y &= f(u) \end{aligned}\right\} u \in [a,b]$	$\left.\begin{aligned} x &= f(u) \\ y &= f(u) \\ z &= f(u) \end{aligned}\right\} u \in [a,b]$
Arc Length Parametric	$\left.\begin{aligned} x &= f(s) \\ y &= f(s) \end{aligned}\right\} s \in [a,b]$	$\left.\begin{aligned} x &= f(s) \\ y &= f(s) \\ z &= f(s) \end{aligned}\right\} s \in [a,b]$
Curvature/Torsion Profile	$\kappa = f(s)$ $\tau = 0$	$\kappa = f(s)$ $\tau = f(s)$

Table 2.1. Curve Representations

	Advantages	**Disadvantages**
Implicit	• Good mathematical understanding of singularities	• Becomes increasingly complex as curve degree gets larger

	(double points, cusps, etc) • Historical literature and research	• Difficult to represent in spatial form • Difficult to describe an actual motion along its arc length
Standard Parametric	• Provides a one-to-one mapping from $R \rightarrow R^3$ • Easy to define in a finite interval as for piecewise segments • Easy to define in spatial form	• Lack of physical meaning in term of the independent parameter • Some loss of mathematical understanding compared to implicit forms
Arc Length Parametric	• Provides good physical meaning to independent parameter • Easy to define physical motion along curve • Calculation of some curve properties becomes easier	• Difficult to find closed-form solutions for most curves • Numerical techniques needed
Curvature/ Torsion Profile	• Defines curve based on higher-order properties • Geometric shape is independent of position/orientation	• Difficult to define global motions • Best used for defining local geometry

Table 2.2. Comparison of Curve Representations

As Table 2.2 shows, each representation for algebraic curves has its advantages and disadvantages depending on the specific application. Standard parametric form is the

easiest representation to deal with and the representation most often used in current literature and applications. However, it is important to understand and preserve the mathematical understanding provided by some of the other curve representations. In the course of this research, we will attempt to provide descriptions of curves and shapes using as many representations as possible.

3. CHAPTER THREE

Interactive Curve Design

In the last chapter, the basic mathematics of algebraic curves was introduced. This chapter will focus on methods of interactive curve design that build on this mathematics. The majority of these methods come from the disciplines of Computer Graphics and Computer-Aided Design. While some of these methods may not be directly applicable to motion planning for physical systems, it is an important starting point in evaluating methods for operator-defined path plans. The first section will describe the need for piecewise curves and introduce the concept of continuity. Next, a few basic but important curve designing schemes will be described: Bezier curves and B-Splines. Then, a few more complex methods that build on these methods will be introduced. It should be noted that these methods will presented only in terms of their basic formulations and results. For more complete derivations of these methods, the provided references can be consulted. Finally, these methods will be evaluated based on their application to manipulator motion planning.

3.1. INTRODUCTION TO PIECEWISE CURVES

Curve design involves generating a mathematical description of a curve (or set of curves) that satisfies a set of given constraints. In its simplest form, this involves generating a curve that passes through a set of defined points. However, to meet n position constraints, an n-1 degree curve would be required. This may be adequate for meeting a small number of constraints, but high-degree polynomials will lead to undesirable behavior such as overshoots and oscillations. This is shown in Figure 3.1 for a curve defined to pass through seven points.

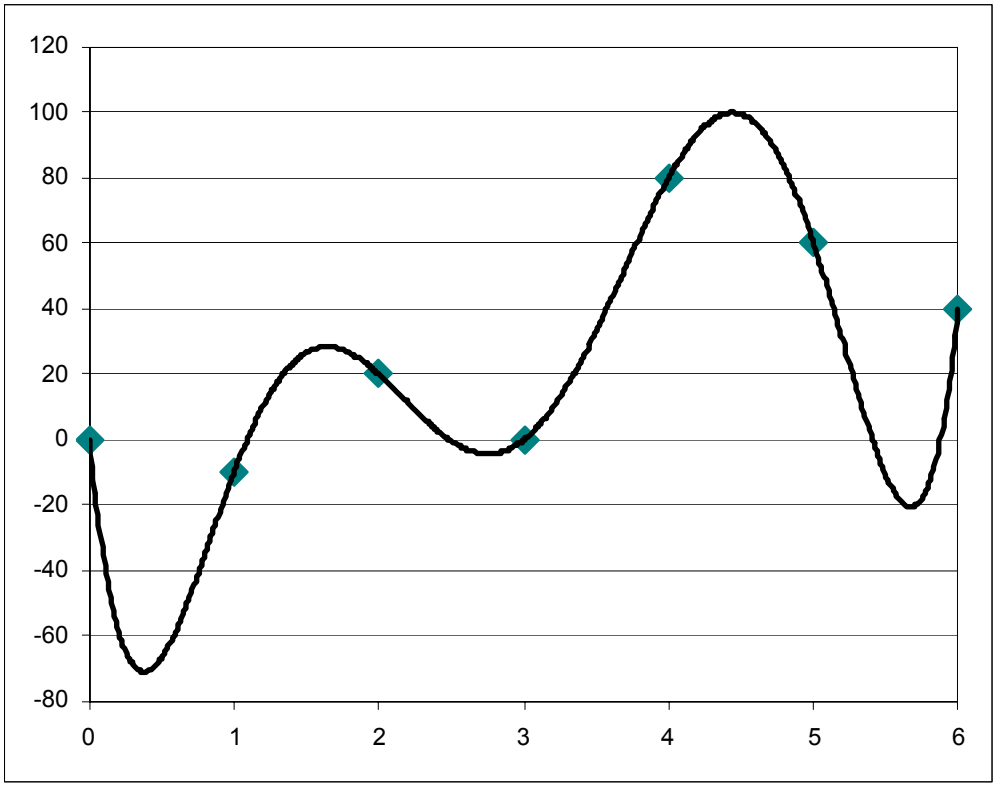


Figure 3.1. High Degree Polynomial Curve

To get around this problem, the shape of the entire curve is usually broken into smaller segments. This allows for the desired constraints to be met with a set of smaller degree curves. For example, each segment between two successive points could be defined as a simple cubic curve as shown in Equation 3.1. This would allow an additional first derivative constraint to be defined at the start and end of each segment. These curves segments can then be pieced together to form the overall shape of the curve. An important concept in piecewise curves such as this is *continuity*, which is basically a measure of the smoothness of the transition between segments. This concept will be introduced in the following section.

$$p(u) = a_0 + a_1 u + a_2 u^2 + a_3 u^3 \qquad \textbf{3.1}$$

3.1.1. Continuity

As mentioned above, continuity is basically a measure of the smoothness of a piecewise curve at its joining points. There are several different notions of continuity. In this section, three types will be defined: Parametric, Geometric, and Frenet Frame.

3.1.1.1. Parametric

The simplest, but most restrictive, type of continuity is known as Parametric Continuity, denoted as C^n for n^{th} order continuity. Two parametric curves meet with C^n continuity at a point if the n^{th} derivative (and all lower derivatives) are exactly equal. For example, consider a curve defined by a set of curve segments $\left(\mathbf{p}_0(u), \mathbf{p}_1(u), \cdots, \mathbf{p}_n(u)\right)$ all normalized such that $u \in [0,1]$. The i^{th} join point is said to be C^0 continuous (i.e. position continuous) if $\mathbf{p}_i(1) = \mathbf{p}_{i+1}(0)$ and C^1 continuous if $\frac{d\mathbf{p}_i(1)}{du} = \frac{d\mathbf{p}_{i+1}(0)}{du}$, etc. In Figure 3.2, the curve segments on the left meet with C^0 continuity, and the curve segments on the right meet with C^1 continuity.

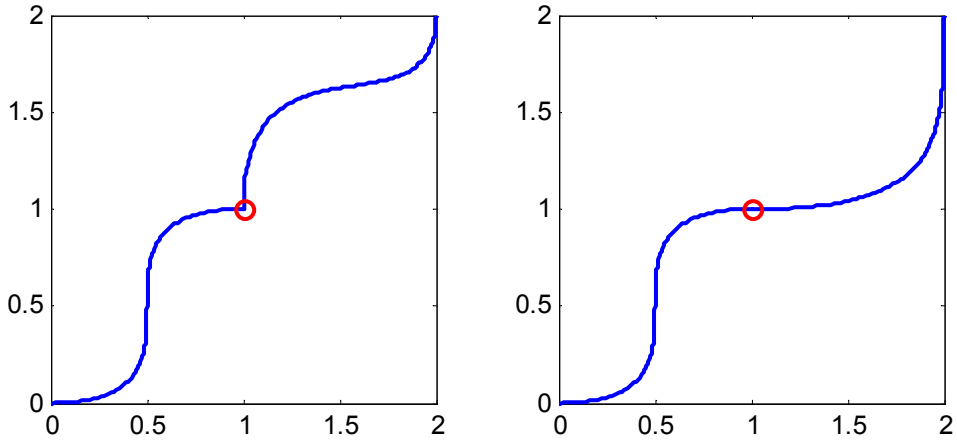


Figure 3.2. C^0 and C^1 Continuous Piecewise Curves

3.1.1.2. Geometric

Geometric continuity (denoted as G^n) is a less restrictive notion of continuity that measures the smoothness of a curve's intrinsic properties at join points. An intrinsic

property, as defined by Barsky and DeRose in [7], is defined as a property that is shared by all equivalent parameterizations of a curve. To understand this, consider the two parametric lines shown in Equation 3.2.

$$\begin{aligned} \mathbf{p}_1(u) &= (2u, u), u \in [0,1] \\ \mathbf{p}_2(v) &= (4v+2, 2v+1), v \in [0,1] \end{aligned} \qquad \textbf{3.2}$$

It should be clear that these curve segments meet with C^0 continuity ($\mathbf{p}_1(1) = \mathbf{p}_2(0)$) but not C^1 continuity ($\mathbf{p}_1^1(1) = [2,1], \mathbf{p}_2^1(0) = [4,2]$). However, it is easy to see that the unit tangent vector is continuous even though the first derivative vector is not, because the lines are collinear. Thus, the unit tangent vector is an intrinsic property and the curves are still considered G^1 continuous even though there is a jump in the first derivative vector. Likewise, G^2 continuity is a measure of smoothness in curvature rather than just the second derivative vector.

Geometric continuity is popular in the field of Computer Graphics because it will lead to visually smooth curves even if the parametric curves (and their higher derivatives) are not equal. This would appear to make this concept inadequate for planning motions for physical systems where jumps in the higher derivatives can lead to undesired shocks in the system. However, this is not necessarily true. As mentioned in the last chapter, the best way to study or define the physical motion along a curve is to look at curves that are parameterized by arc length[2], and a property of arc length parameterized curves is that parametric continuity and geometric continuity are equivalent. This means that if a curve is transversed at some constant or smoothly defined speed, it is the intrinsic properties that define the smoothness of the motion. Thus, in terms of defining a physical motion along a curve, geometric continuity actually provides better physical meaning as well as being less restrictive.

[2] A method for defining motions along standard parametric curves was also introduced.

3.1.1.3. Frenet Frame

The last notion of continuity explored here is Frenet Frame continuity, denoted as F^n for n^{th} order continuity. Frenet Frame continuity is based on the continuity of a curve's generalized curvatures [22]. For a curve in R^3 with arc length parameterization, the unit tangent vector can be written as $\mathbf{t}_1(s) = \mathbf{p}^{(1)}(s)$. From the relationship $\hat{\mathbf{N}} = \frac{1}{\kappa}\frac{d^2\mathbf{p}}{ds^2}$, the unit normal can be written as $\mathbf{t}_2(s) = \frac{\mathbf{t}_1^{(1)}(s)}{\kappa_1(s)}$ where $\kappa_1(s)$ is the curvature. Finally, from the Frenet-Serret formulae the unit bi-normal can be written as $\mathbf{t}_3(s) = \frac{\mathbf{t}_2^{(1)}(s) + \kappa_1(s)\mathbf{t}_1(s)}{\kappa_2(s)}$ where $\kappa_2(s)$ is the torsion. Here, $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ are the generalized curvature vectors in R^3. Thus, in R^3, F^1 continuity is continuity of the unit tangent, F^2 relates to continuity of the unit normal, and F^3 relates to continuity of the bi-normal vector. For a curve lying in R^d, the generalized curvatures $(\mathbf{t}_1(s), \mathbf{t}_2(s), \cdots, \mathbf{t}_d(s))$ can be found using Equation 3.3.

$$\begin{aligned} \mathbf{t}_1(s) &= \mathbf{p}^{(1)}(s) \\ \kappa_0(s) &= 0 \\ \mathbf{t}_{i+1} &= \frac{\mathbf{t}_i^{(1)}(s) + \kappa_{i-1}(s)\mathbf{t}_{i-1}(s)}{k_i(s)} \end{aligned} \qquad \textbf{3.3}$$

Frenet Frame continuity is often used for defining curves in higher dimensions. However, it is very similar to geometric continuity for lower dimension curves. This is because G^n and F^n continuity is equivalent for n=1 and n=2. Also, a curve in R^d that is F^d will also be trivially F^i for all $i \geq d$ [22].

3.2. BASIC SPLINE TECHNIQUES

In this section, two simple curve design techniques will be introduced: Bezier Curves and B-Splines. The basic mathematical constructions and properties of these

curves will be discussed. Some of the material developed here is also used later as a framework for more complicated curve generation schemes.

3.2.1. Bezier Curves

A Bezier Curve is a classic curve design technique that is specified by a set of control vertices (sometimes called a control polygon or scaffold). The curve will interpolate the first and last point on the polygon and will follow the general shape of the polygon in a smooth fashion. Figure 3.3 shows an example Bezier curve where the dotted lines represent the control polygonal and the solid line represents the resulting curve.

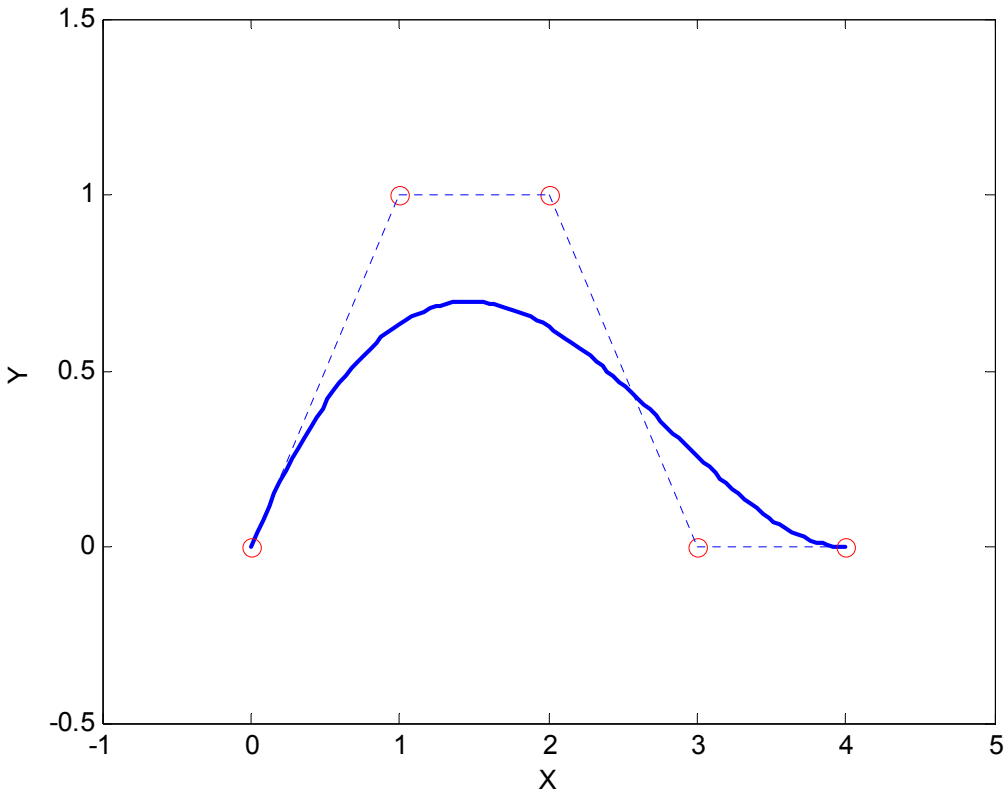


Figure 3.3. Example Bezier Curve

3.2.1.1. Formulation

The basic formulation for a parametric Bezier Curve is shown in Equation 3.4 where d is the degree of the curve, $\mathbf{b}_i$ are the control vertices, and B_i are the Bernstein basis polynomials (shown in Equation 3.5 for one dimension). The Bernstein basis polynomials are an alternative to the standard power basis $\left(1, u, u^2, \cdots, u^d\right)$ and have the property of *partition of unity* (the basis polynomials sum to unity at any point over the range).

$$\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(u), u \in [0,1] \quad \textbf{3.4}$$

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i} = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \quad \textbf{3.5}$$

From the above equations, it can be seen that for a control polygon with d vertices the degree of the curve will be d-1. For example, we can expand Equation 3.4 and write the parametric description of a cubic curve as shown in Equation 3.6. Additionally, each individual Bernstein polynomial can be plotted as shown in Figure 3.4.

$$\begin{aligned} \mathbf{p}(u) &= B_0\mathbf{b}_0 + B_1\mathbf{b}_1 + B_2\mathbf{b}_2 + B_3\mathbf{b}_3 \\ &= (1-u)^3 \mathbf{b}_0 + 3u(1-u)^2 \mathbf{b}_1 + 3u^2(1-u)\mathbf{b}_2 + u^3\mathbf{b}_3 \end{aligned} \quad \textbf{3.6}$$

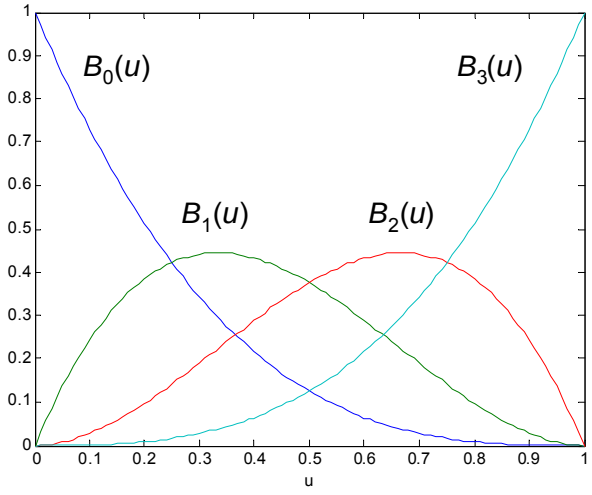


Figure 3.4. Bernstein Basis Polynomials

From this plot, the value of B_0 at u=0 is 1, and the value of B_3 at u=1 is 1. This means the curve will interpolate the control points $\mathbf{b}_0$ and $\mathbf{b}_3$. Also, it should be noted that with the exception of the end points, all of the Bernstein polynomials are non-zero throughout the range. This means that moving any point on the control polygon will affect the shape of the entire curve.

3.2.1.2. Properties

Bezier Curves have several important properties that deserve mention here:

- Convex Hull Property
- Variation-Diminishing Property

The convex hull property states that the entire curve will lie inside the convex hull of the control polygon. This provides a simple bounding-box that can be used to constraint the boundaries of the curve. The Variation-Diminishing Property basically states that the resulting Bezier Curve is as well-behaved as its control polygon. More specifically, a given line will intersect the curve at no more points than it will intersect the control polygon. This means that even for higher-degree Bezier Curves there will not be any unpredictable oscillations as with standard polynomials.

3.2.1.3. Example

Figure 3.5 shows two example Bezier Curves. The curve on the left is a simple degree 3 curve. The curve on the left is of degree six with seven specified vertices. Despite the higher degree, the curve is still well-behaved inside the control polygon.

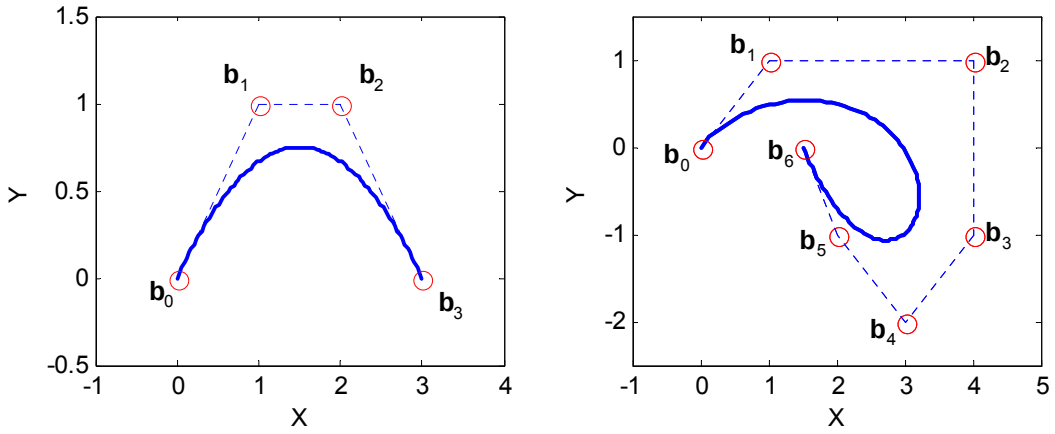


Figure 3.5. Two Example Bezier Curves

This section has so far focused on planar curves because they are easier to visualize. However, the Bezier Curve formulation is extendable to any number of dimensions. Figure 3.6 shows a helical shaped spatial Bezier Curve.

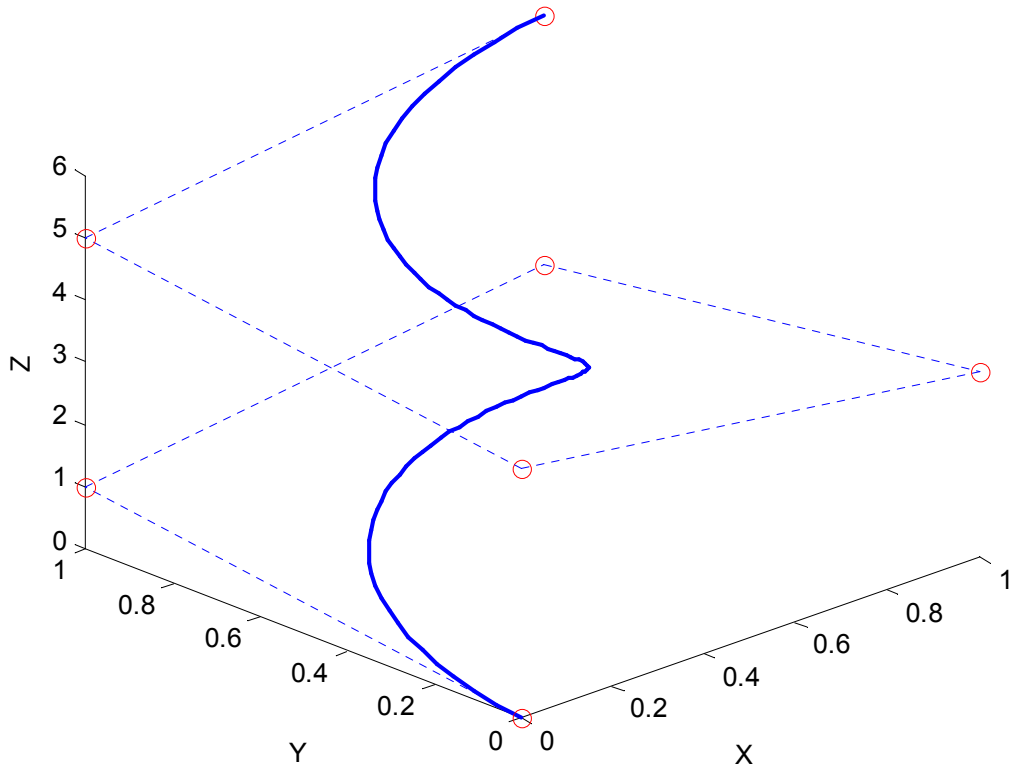


Figure 3.6. Spatial Bezier Curve

3.2.2. B-Splines

3.2.2.1. Formulation

B-Splines are specified in much the same way as Bezier Curves by defining a control polygon. Given a set of control points $[\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n]$, a B-Spline is defined as shown in Equation 3.7 where $N_{i,k}$ are the B-Spline basis functions. Unlike in a Bezier Curve where the degree of the curve is determined by the number of control points, the degree can be specified in a B-Spline and is equal to k-1.

$$\mathbf{P}(t) = \sum_{i=1}^{n} \mathbf{b}_i N_{i,k}(u) \qquad \textbf{3.7}$$

The B-Spline basis functions are defined recursively as shown in Equation 3.8. In the special case that k=1, the basis function is defined in Equation 3.9. This condition stops the recursion. The u_i values referenced in the above equation represent values in a B-Spline knot vector of size $n+k$. In general, a B-Spline does not interpolate any of its control vertices. However, the knot vector can be formulated in such a way that the curve interpolates the first and last end point as in a Bezier Curve. This involves repeating values at the beginning and end of the vector to force interpolation. Figure 3.7 shows a simple B-Spline curve along with its knot vector.

$$N_{i,k}(u) = \frac{N_{i,k-1}(u)(u - u_i)}{(u_{i+k-1} - u_i)} + \frac{N_{i+1,k-1}(u)(u_{i+k} - u)}{(u_{i+k} - u_{i+1})} \qquad \textbf{3.8}$$

$$N_{i,1} = \begin{Bmatrix} 1, \; u \in [u_i, u_{i+1}] \\ 0, \text{ otherwise} \end{Bmatrix} \qquad \textbf{3.9}$$

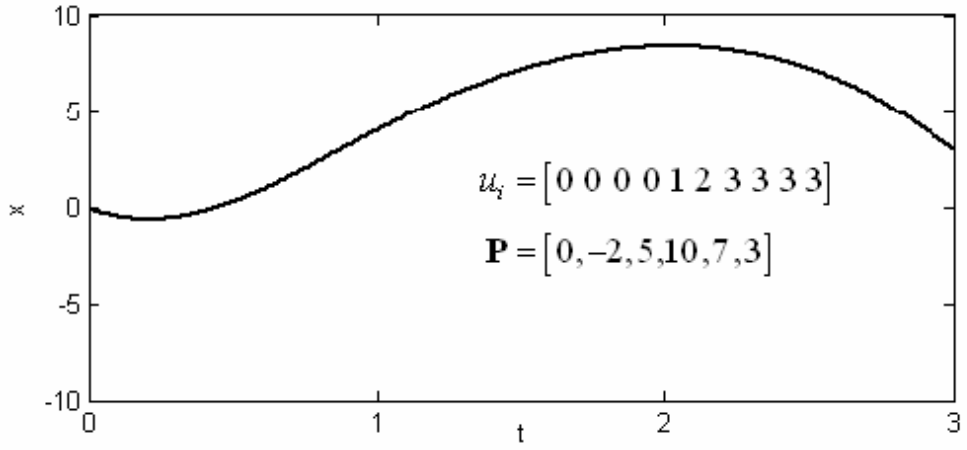


Figure 3.7. B-Spline along with Knot Vector

3.2.2.2. Local Control Property

In addition to the variation-diminishing and convex hull properties, B-Splines also exhibit *local control*. As shown in Equation 3.9, each of the B-Spline basis functions is only non-zero for a specific range. This means that moving the control vertices will only affect the shape of the B-Spline in the local area around that point. This can be seen by plotting the individual basis functions as in Figure 3.8.

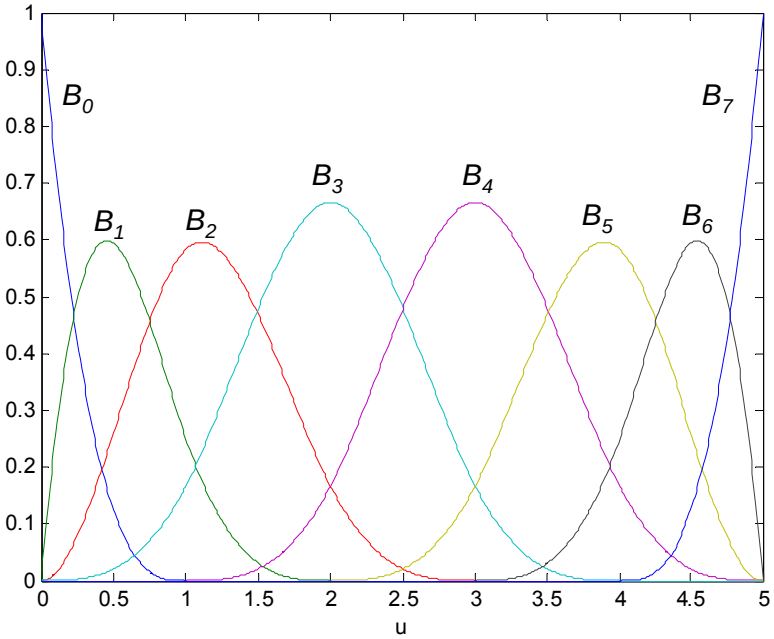


Figure 3.8. B-Spline Basis Functions

3.2.2.3. Example

Figure 3.9 shows two different degree 3 B-Splines defined by eight control points. They both share the first six points with the last two differing. This plot shows the property of local control as both curves are identical until the last portion.

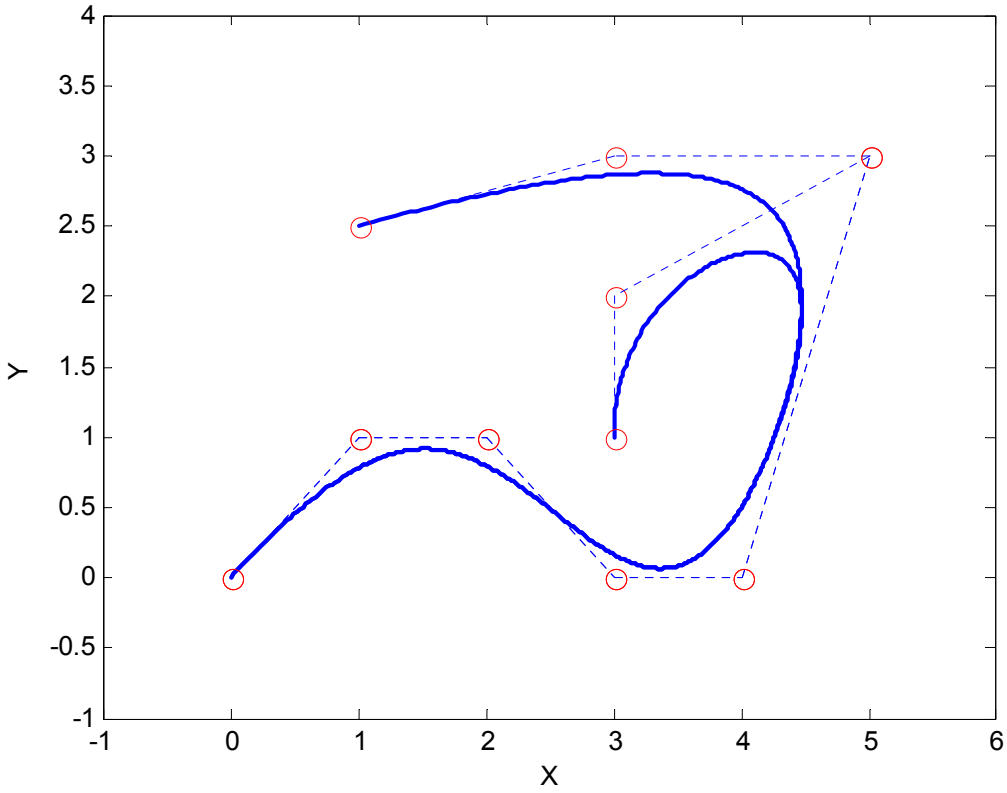


Figure 3.9. Two Example B-Spline Curves

3.3. BETA-SPLINES

Beta-splines are geometrically continuous piecewise curves based on the concept of *Beta-constraints* (also called *shape parameters*) [9][14]. These curves use the extra degrees of freedom gained from relaxing parametric continuity to geometric continuity as design parameters. This section will first demonstrate how to build piecewise Bezier Curves with parametric continuity. Then, the concept of Beta-constraints will be introduced, and it will be shown how these can be used to create more options in curve

design. Only the basics of these formulations will be introduced in this section. More detail on the theory and derivation of these curves can be found in [10].

3.3.1. Parametric Continuity for Bezier Curves

This section will briefly introduce how to form parametrically continuous (C^n) piecewise Bezier Curves. Consider two Bezier curves, $\mathbf{P}_0(u)$ and $\mathbf{P}_1(u)$, defined by control vertices $\left[\mathbf{b}_{0,1}, \mathbf{b}_{0,2}, \cdots, \mathbf{b}_{0,d}\right]$ and $\left[\mathbf{b}_{1,0}, \mathbf{b}_{1,1}, \cdots, \mathbf{b}_{1,d}\right]$, respectively. Now, parametric continuity needs to be enforced at the at the join point between these two segments ($\mathbf{b}_{0,d} = \mathbf{b}_{1,0}$). A sketch of these two control polygons meeting is shown in Figure 3.10.

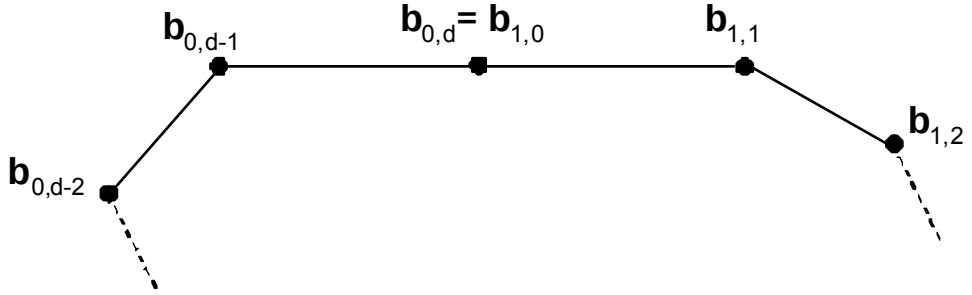


Figure 3.10. Two Joined Bezier Control Polygons

For C^0 continuity, it is obvious that the last vertex of the first polygon must be equal to the first vertex of the second polygon because the two segments will interpolate these points. It can be shown that the first derivatives of these two curves can be written in terms of their control vertices as shown in Equation 3.10. This relationship shows that the last edge of the first polygon and the first edge of the second polygon must be collinear and equal length to enforce C^1 continuity.

$$\begin{aligned} \mathbf{P}_0^{(1)}(1) &= d\left(\mathbf{b}_{0,d} - \mathbf{b}_{0,d-1}\right) \\ \mathbf{P}_1^{(1)}(0) &= d\left(\mathbf{b}_{1,1} - \mathbf{b}_{1,0}\right) \end{aligned} \qquad \textbf{3.10}$$

Similarly, the second derivative can be written in terms of the control vertices as shown in Equation 3.11. This shows that to enforce C^2 continuity constraints must be applied to the three vertices closest to the join point. However, since the the two closest points are already constrained to enforce C^0 and C^1 continuity, this becomes a constraint

on the points $\mathbf{b}_{0,d-2}$ and $\mathbf{b}_{1,2}$. This means that for two cubic Bezier curves to meet with C^2 continuity all interior control vertices become constrained. As the order of continuity desired goes up, constraints must be applied to additional vertices. Later in this section, it will be shown that by relaxing to geometric continuity (which is a more intrinsic form) more degrees of freedom can be attained.

$$\begin{aligned}\mathbf{P}_0^{(2)}(1) &= d(d-1)\left(\mathbf{b}_{0,d-2} - 2\mathbf{b}_{0,d-1} - \mathbf{b}_{0,d}\right)\\ \mathbf{P}_1^{(1)}(0) &= d(d-1)\left(\mathbf{b}_{1,0} - 2\mathbf{b}_{1,1} - \mathbf{b}_{1,2}\right)\end{aligned} \qquad \textbf{3.11}$$

3.3.2. Geometric Continuity for Bezier Curves

An important concept relating to geometric continuity and the construction of Beta-splines is Beta-constraints. These are constants that can be used to determine if curve segments meet with G^n continuity. The Beta-constraint equation for G^1 continuity is shown Equation 3.12 where β_1 is greater than zero to retain directionality. This just shows that the first derivative vectors can be scaled by some constant value while still having the same unit tangent vector (i.e. heading). The Beta-constraint equations for G^2 and G^3 continuity are shown Equations 3.13 and 3.14. Because β_1 can be defined to be any positive number, this actually leads to some design choices in how to define the curve segments while still keeping geometric continuity.

$$\beta_1 \mathbf{p}_i^{(1)}(1) = \mathbf{p}_{i+1}^{(1)}(0) \qquad \textbf{3.12}$$

$$\beta_1^2 \mathbf{p}_i^{(2)}(1) + \beta_2 \mathbf{p}_i^{(1)}(1) = \mathbf{p}_{i+1}^{(2)}(0) \qquad \textbf{3.13}$$

$$\beta_1^3 \mathbf{p}_i^{(3)}(1) + 3\beta_1\beta_2 \mathbf{p}_i^{(2)}(1) + \beta_3 \mathbf{p}_i^{(1)}(1) = \mathbf{p}_{i+1}^{(3)}(0) \qquad \textbf{3.14}$$

Now, a general algorithm for using these constraints to design geometrically continuous Bezier curves can be developed. The details and derivation of this method can be found in [10]. First, start with one segment of the curve defined by a set of control vertices $\left[\mathbf{b}_{0,1}, \mathbf{b}_{0,2}, \cdots, \mathbf{b}_{0,d}\right]$. Now, the constrained vertices of the second control polygon

can be determined by the steps shown in Equation 3.15 where β_1 and β_2 are user-defined design parameters. Figure 3.11 shows two cubic Bezier curves meeting with G^2 continuity for varying values of β_2.

$$\gamma = \frac{(d-1)(1+\beta_1)}{\beta_2 + \beta_1(d-1)(1+\beta_1)}$$
$$\mathbf{b}_{1,0} = \mathbf{b}_{0,d}$$
$$\mathbf{b}_{1,1} = \mathbf{b}_{1,0} + \beta_1\left(\mathbf{b}_{0,d} - \mathbf{b}_{0,d-1}\right) \qquad \textbf{3.15}$$
$$\mathbf{T} = \mathbf{b}_{0,d-1} + \beta_1^2\gamma\left(\mathbf{b}_{0,d-1} - \mathbf{b}_{0,d-2}\right)$$
$$\mathbf{b}_{1,2} = \mathbf{b}_{1,1} + \frac{1}{\gamma}\left(\mathbf{b}_{1,1} - \mathbf{T}\right)$$

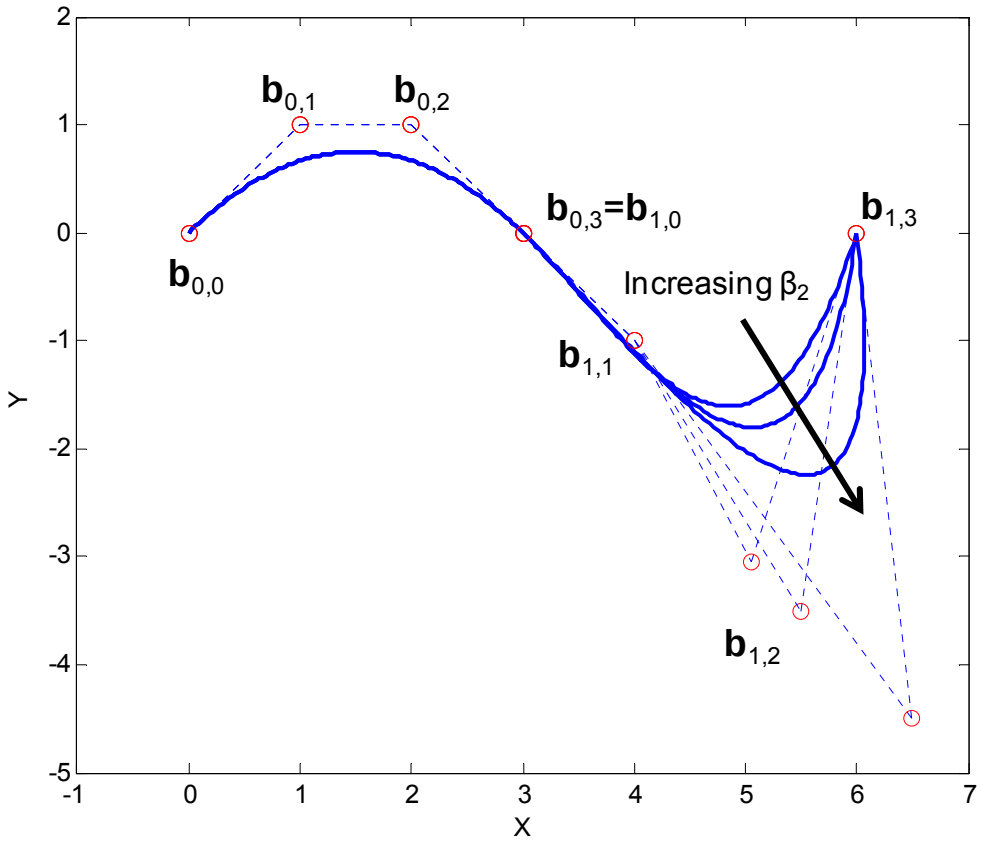


Figure 3.11. Geometrically Continuous Bezier Curves

3.3.3. Beta-spline Formulation

Beta-splines build on the concepts of geometric continuity and beta-constraints to offer even more options for designing a curve. This section will describe one method of creating G^2 continuous cubic Beta-Splines [9]. However, it should be noted that these concepts can be expanded for more complicated curves [7][14]. A Beta-spline is defined once again by a set of control vertices as well as set of β_1 and β_2 values. The original control polygon is then divided into several interior control polygons as shown in Figure 3.12. Then, a set of Bezier curves can be drawn using these subdivided polygons. These interior vertices are calculated with respect to the β_1 and β_2 values in order to ensure G^2 continuity inside the curve. The calculation of these interior vertices is shown in Equation 3.16.

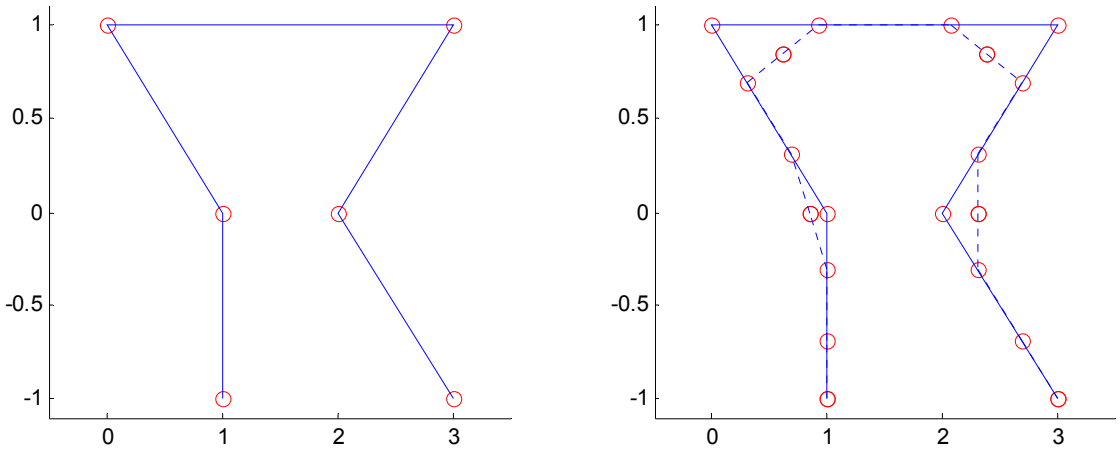


Figure 3.12. Subdivided Control Polygon

$$\gamma_i = \frac{2\left(1+\beta 1_i\right)}{\beta 2_i + 2\beta 1_i\left(1+\beta 1_i\right)}$$

$$\mathbf{W}_{i,1} = \frac{\left(1+\beta 1_{i+1}^2\gamma_{i+1}\right)\mathbf{b}_i + \gamma_i \mathbf{b}_{i+1}}{1+\gamma_i+\beta 1_{i+1}^2\gamma_{i+1}} \qquad \textbf{3.16}$$

$$\mathbf{W}_{i,2} = \frac{\beta 1_{i+1}^2\gamma_{i+1}\mathbf{b}_i + \left(1+\gamma_i\right)\mathbf{b}_{i+1}}{1+\gamma_i+\beta 1_{i+1}^2\gamma_{i+1}}$$

Now, the β_1 and β_2 values can be used as degrees of freedom to change the interior local shape of the curve. Figure 3.13 shows the effect of varying the β_1 parameter on the shape of the interior curve. This shows that, by increasing this value, the curve tends to favor the tangent line coming off the original control vertices. This makes sense, because the β_1 parameter is basically increasing the parametric value of the first derivative. This parameter is often also called *bias*.

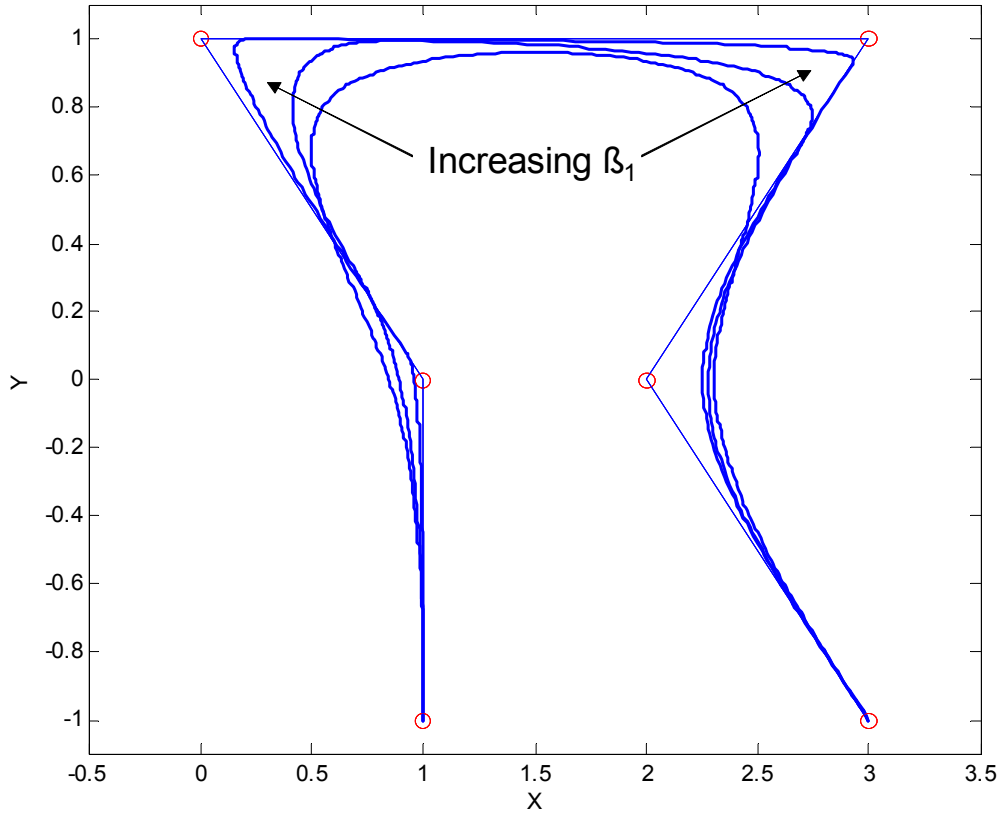


Figure 3.13. Effect of β_1 on the shape of a Beta-spline

Figure 3.14 shows the effect of varying the β_2 parameter on the shape of the interior curve. By increasing this value, the curve tends to more closely follow the original control polygon. This parameter is often also called *tension*.

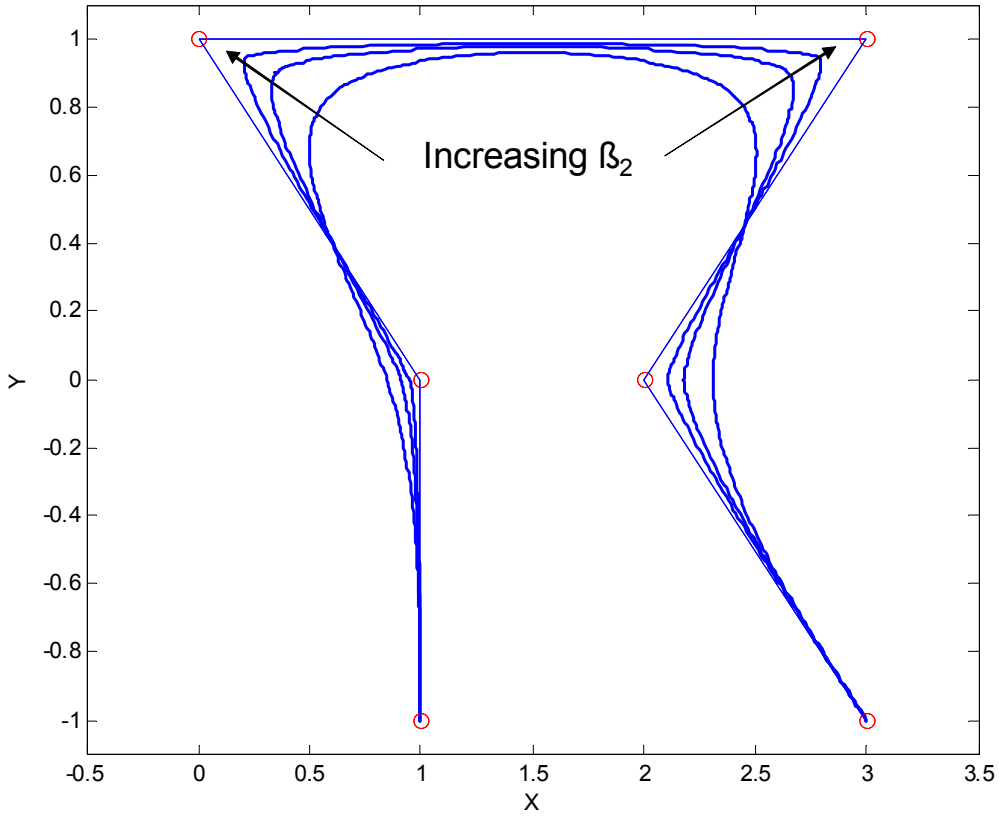


Figure 3.14. Effect of β_2 on the shape of a Beta-spline

3.3.4. Conclusions

Beta-splines provide a way of using the extra degrees of freedom attained by relaxing the continuity of piecewise curves as a design tool. As discussed in Section 3.1.1, geometric continuity is a more natural way of representing the continuity at join points because it focuses on the continuity of the intrinsic properties of curves. In terms of motion along a curve, this is a superior way of defining smoothness. Thus, Beta-splines are a useful formulation to study in terms of curve generation. However, the physical meanings of the control parameters, β_1 and β_2, are difficult to quantify and designing curves in this style would require much trial-and-error.

3.4. ALGEBRAIC SPLINES

A-Splines (Algebraic Splines) are real algebraic curve segments that are defined in tensor Bernstein-Bezier form [5]. These splines are defined as the zero contour of a function defined in a triangular polygon in barycentric coordinates and allow for a higher degree of geometric continuity while still maintaining some degrees of freedom for curve design. This section will describe the basic formulation of A-Splines and how they can be applied to curve design.

3.4.1. Barycentric Coordinates

Barycentric coordinates defined on a triangle provide a local coordinate system defined with respect to the vertices of the triangle. Consider the triangle shown in Figure 3.15. The trilinear set of barycentric coordinates are defined as shown in Equation 3.17. This shows that the three coordinates are ratios of the areas of the interior triangles defined by the point **p**. These coordinates also have the constraints that $\alpha_i \leq 1$ and $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

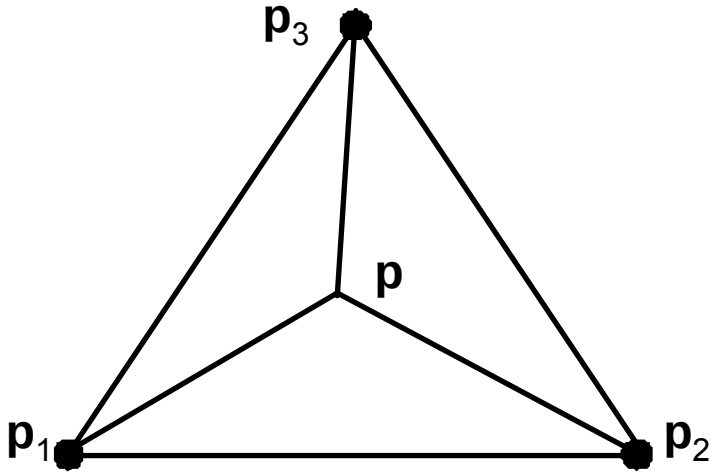


Figure 3.15. Triangular Coordinate System

$$(\alpha_1, \alpha_2, \alpha_3) = \left(\frac{area(\mathbf{p}, \mathbf{p}_2, \mathbf{p}_3)}{area(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)}, \frac{area(\mathbf{p}, \mathbf{p}_1, \mathbf{p}_3)}{area(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)}, \frac{area(\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2)}{area(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)} \right) \quad \textbf{3.17}$$

The barycentric coordinates can be mapped back to x and y coordinates as shown in Equation 3.18. From this mapping, it is easy to see that $\alpha_i = 1$ translates to point p_i.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \quad \textbf{3.18}$$

3.4.2. Formulation

An A-Spline can now be defined in Bernstein-Bezier form as shown in Equation 3.19 where $B_{ijk}^n(\alpha_1,\alpha_2,\alpha_3) = \frac{n!}{i!\,j!\,k!}\alpha_1^i\alpha_2^j\alpha_3^k$. The b_{ijk} coefficients are scalar quantities defined at points on the triangle. The number of coefficients depends on the degree of the A-Spline. For example, a cubic A-spline would have the Bezier coefficients as shown in Figure 3.16. Then, the curve is defined as the set of all $(\alpha_1,\alpha_2,\alpha_3)$ that provide zero values from Equation 3.19.

$$F(\alpha_1,\alpha_2,\alpha_3) = \sum_{i+j+k=n} b_{ijk} B_{ijk}^n(\alpha_1,\alpha_2,\alpha_3) = 0 \quad \textbf{3.19}$$

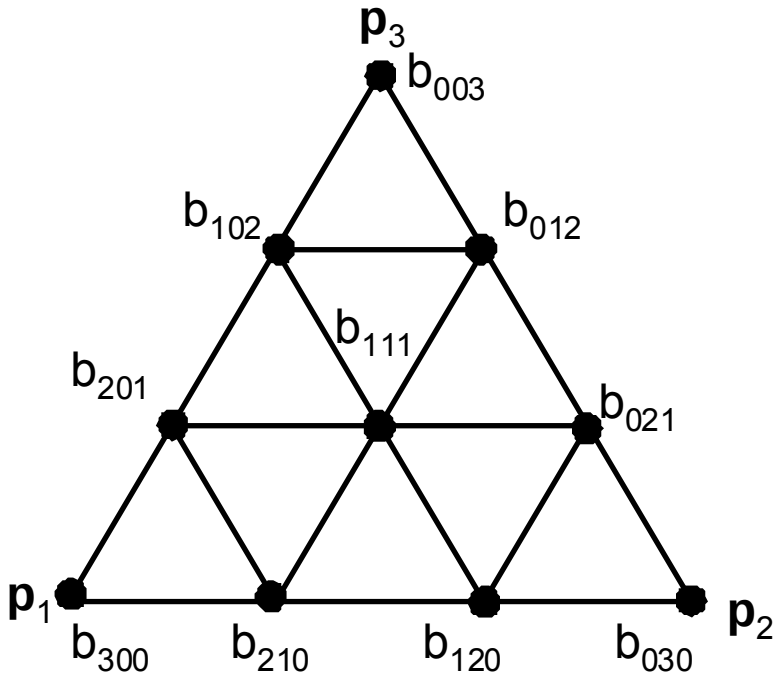


Figure 3.16. Bezier Coefficients for a Cubic A-Spline

In order to find a solution to this equation, the following constraint must first be applied to the b_{ijk} coefficients: there must be one and only one sign change on the coefficients running along the $\mathbf{p}_1\mathbf{p}_3$ and $\mathbf{p}_2\mathbf{p}_3$ line segments. Basically, this ensures that the resulting curve will intersect both of these line segments once and only once. Then, we can take the family of lines of the form $(\alpha_1,\alpha_2,\alpha_3)(t)=(1-t)(\beta,1-\beta,0)+t(0,0,1)$ for $\beta\in[0,1]$. This is the family of lines running from $\mathbf{p}_3$ to the line segment $\mathbf{p}_1\mathbf{p}_2$. Substituting this into the original formulation (Equation 3.19), we arrive at Equation 3.20. For degree $d\le 4$ curves, a closed form solution to this equation can be found. For higher degree curves, this equation can be solved with a simple root-finding technique.

$$B_\beta(t)=\sum_{i+j+k=n} b_{ijk}\frac{n!}{i!\,j!\,k!}t^k(1-t)^{i+j}\beta^i(1-\beta)^j=0 \qquad \textbf{3.20}$$

3.4.3. Piecewise A-Splines

A-Splines are often used to interpolate data points or approximate polygonal chains. One advantage of A-Splines over analogous techniques is that they can in general achieve G^{2d-3} [5] continuity while still maintaining some degree of freedom over the shape of the curve. Basically, constraints can be set on the b_{ijk} coefficients such that the desired conditions are met at the join points of the curve segments. Then, the unconstrained coefficients can be used to alter the shape of the curve.

A simple example of this is designing G^1 continuous (unit tangent continuous) piecewise A-Splines. It can be easily shown that by setting $b_{300}=b_{030}=0$ the curve will interpolate $\mathbf{p}_1$ and $\mathbf{p}_2$. Similarly, by setting $b_{201}=b_{021}=0$, the curve will be tangent to the lines $\mathbf{p}_1\mathbf{p}_3$ and $\mathbf{p}_2\mathbf{p}_3$. In order to maintain one sign change along the edges, we additionally constraint $b_{102},b_{012},b_{003}<0$ and $b_{201},b_{120}>0$. The coefficient b_{111} is a completely free parameter. Thus, G^1 continuity can be achieved by lining up the edges of adjacent triangles, and the unconstrained and semi-constrained values can be used to generate

varying families of curves within each segment. An example of this for varying b_{111} values is shown in Figure 3.17.

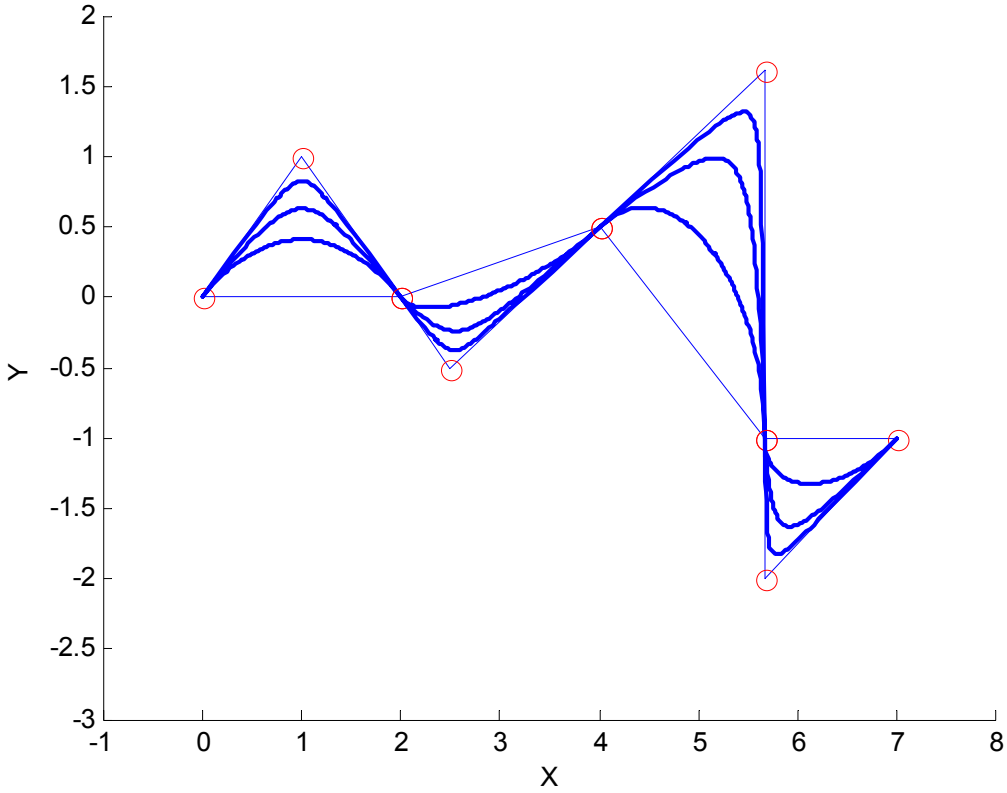


Figure 3.17. Families of G^1 Continuous A-Splines

3.4.4. Implicit Representations

Another interesting application for A-Splines is in representing implicit forms of curves ($f(x,y)=0$). Because an A-Spline itself is an implicit equation in barycentric coordinates, they are able to capture this representation naturally. This can be done by inverting the relationship given in Equation 3.18 to come up with the relationships $(\alpha_1,\alpha_2,\alpha_3)=f(x,y)$. Then, these equation can be plugged into Equation 3.19, and the b_{ijk} coefficients can be determined such that the original polynomial in x and y is recovered.

For example, take the cusp $y^2 - x^3 = 0$ discussed in Chapter Two. First a triangular scaffolding around the area of interest is constructed as shown in Figure 3.18. For this curve, two triangles are used to avoid having a singular point inside of the triangle. Now, the coefficients can be calculated as $b_{003} = -1$, $b_{012} = -1$, $b_{021} = -\frac{2}{3}$, $b_{012} = \frac{1}{3}$, and all other coefficients zero. A full derivation of this result can be found in Appendix A.

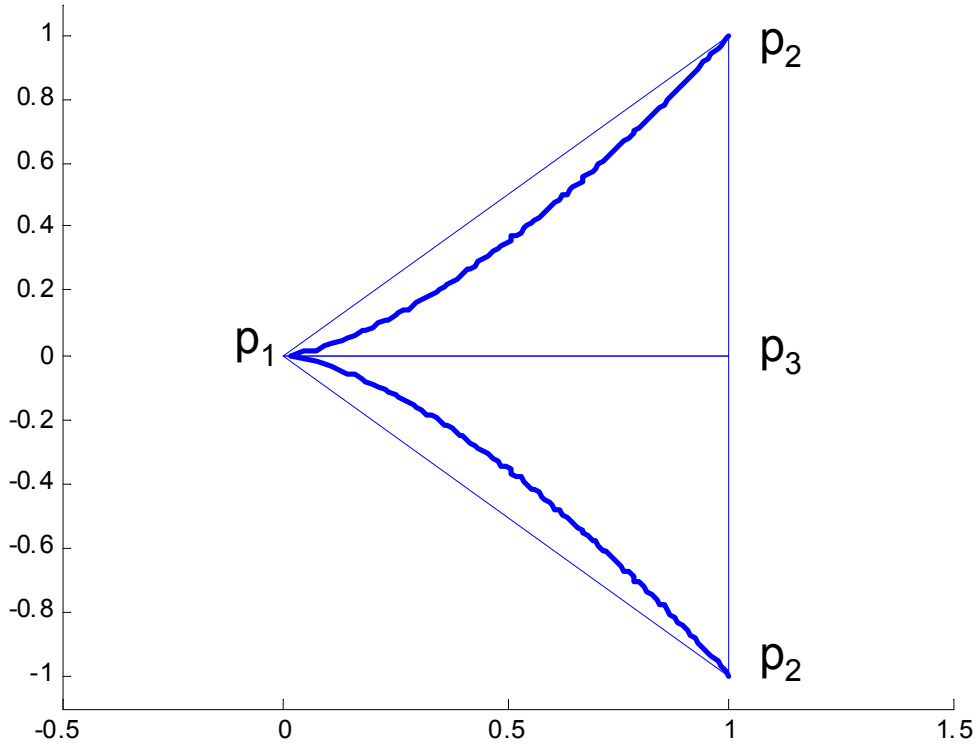


Figure 3.18. A-Spline Representation of a Cusp

There are many advantages of capturing implicit forms of curves in this manner. As mentioned in Chapter 2, implicit forms of curves are very good at providing mathematical insight. However, they are difficult to actually define a motion along and must be converted (when possible) to parametric forms. By defining them in this manner,

the curves can be easily traced while still taking advantage of the mathematical meaning of the implicit representation. Also, by describing the curve based on its local geometry inside a triangle, the curve can easily be translated or rotated to different locations while maintaining the same intrinsic properties.

3.4.5. Conclusions

This section discussed the basic formulations for Algebraic Splines and their applications. In general, A-Splines provide a higher degree of geometric continuity, and thus more degrees of freedom, than the other curve generation techniques discussed in this chapter. Additionally, they provide methods for capturing implicit descriptions of curves.

3.5. SUMMARY

The review of curve generation techniques performed in this chapter is by no means exhaustive. However, this subset of techniques provides an overview into how interactive curve generation has been traditionally approached. First, the desired constraints are defined (position, tangent, curvature or 2^{nd} derivative, etc). Then, a curve is developed to meet these constraints, and the extra degrees of freedom are identified and quantified as design parameters.

The approach taken in this research is similar. However, the emphasis in this research is the actual definition of the constraints rather than the design of the curve between these constraints. This is relevant in the field of robotics as it is important to define intuitive constraints with well-defined physical meanings (e.g. curvature). However, the techniques described in this chapter may be applicable to the process of blending between sets of constraints or even helping to define these constraints.

4. CHAPTER FOUR

Geometric Shapes and Properties: Physical Meaning

4.1. INTRODUCTION

In Chapter Two, the necessary mathematical background for understanding the various properties and representations of curves was introduced. Then, Chapter Three described a variety of methods for interactive curve design and demonstrated a need for an ability to define curve constraints that have more physical meaning. This is important in robotic systems where the curve must be defined with constraints that have physical meaning. This chapter will focus on developing clear physical understanding of the higher-order properties of curves (i.e. curvature and torsion). This will start with an examination of simple shapes such as lines and circles and then move into more complex spatial geometries. It will be shown how the properties of these shapes can be changed to generate families of curves that could be useful for path design. The relationships between these families of curves and the properties that define them will provide useful insight into the definition of constraints based on curvature and torsion for path planning. Then, in the next step of this research, these constraints based on curvature and torsion can be converted into constraints on the parametric description of the curve (e.g. $\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^n x}{du^n}, \frac{d^n y}{du^n}, \frac{d^n z}{du^n}$) that can be more easily blended together into a complete path plan.

4.2. LINEAR SHAPES

4.2.1. Straight Line

The simplest geometric shape is a line. A line is a degree 1 curve that can be represented in its implicit form as shown in Equation 4.1. This is often written in the slope-intercept form as $y = mx + b$ where m is the slope of the line and b is the intersection point on the y-axis. Thus, m=0 corresponds to a horizontal line, and m=∞ corresponds to a vertical line. Figure 4.1 shows the effect of varying slope parameter m to generate a family of lines.

$$ax + by + c = 0 \qquad \textbf{4.1}$$

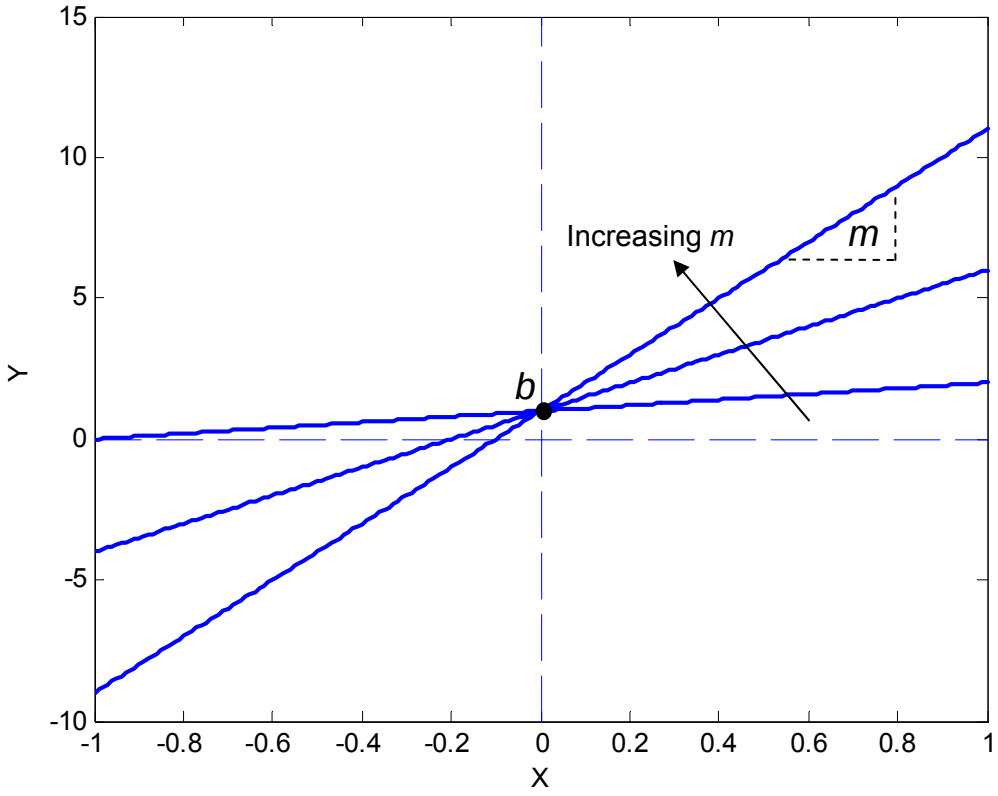


Figure 4.1. Family of Planar Lines

In parametric form, a generalized line can be written as shown in Equation 4.2. If a_1 is set to 1 and b_1 is set to 0, this yields a form that is very similar to the implicit form described above.

$$\begin{aligned} x(u) &= a_1 u + b_1 \\ y(u) &= a_2 u + b_2 \end{aligned} \qquad \textbf{4.2}$$

A more common way to represent a parametric line segment is shown in Equation 4.3. In this form, $\mathbf{p}_i$ and $\mathbf{p}_f$ represent initial and final position vectors of any dimension (e.g. 2 for planar, 3 for spatial). Then, the line will be at $\mathbf{p}_i$ for u=0 and $\mathbf{p}_f$ for u=1. This generalized form is often used to define an interpolation between two points.

$$\mathbf{p}(u) = \mathbf{p}_i + (\mathbf{p}_f - \mathbf{p}_i)u, \; u \in [0,1] \qquad \textbf{4.3}$$

Finally, a line will have a zero curvature and torsion. This was described in Section 2.2.3.1 where curvature was first defined. A zero curvature represents an infinite radius of curvature which translates to a line.

4.3. PLANAR GEOMETRIC SHAPES

This section will begin to explore the properties and representations for simple planar curve shapes. This analysis will begin by defining implicit forms of these shapes and defining parameters that can be used to generate families of curves. Then, closed-form solutions for higher-order properties such as curvature in terms of these parameters will be presented. While manipulator path planning is generally concerned with spatial paths, analysis of simple planar shapes should provide good (and necessary) building blocks towards more complex motions.

4.3.1. Parabola

A parabola is a simple degree 2 curve that can be described by the implicit equation $y - ax^2 = 0$ [3]. It should be noted that a more generalized form of this parabola could be written as $(y - y_c) - a(x - x_c)^2 = 0$ that is centered at the point (x_c, y_c) instead of the origin. However, the higher-order (intrinsic) properties of the curve are our main interest, and these properties remain constant through any translation or rotation of the curve in the reference frame. Thus, for this analysis, it is sufficient to assume these shapes pass through the origin.

Now, a closed-form solution for the curvature of a parabola can be obtained. Equation 4.4 (first introduced in Section 2.1.1) shows the calculation of curvature for an implicitly defined curve. The individual terms in the equation can be easily calculated from the original equation of the parabola as: $f_x = -2ax$, $f_y = 1$, $f_{xx} = -2a$, $f_{yy} = f_{xy} = 0$.

$$\kappa(x, y) = \frac{f_{xx} f_y^2 - 2 f_{xy} f_x f_y + f_{yy} f_x^2}{\left(f_x^2 + f_y^2\right)^{3/2}} \qquad \textbf{4.4}$$

Plugging these values into Equation 4.4, the curvature can be formulated as shown in Equation 4.5. This equation shows that, even for a simple degree 2 curve like a parabola, a closed-form equation for curvature can become complex. However, this equation can still provide some useful insight. Because the quantity $4a^2x^2$ will always be positive, the maximum curvature will always occur at the point x=0 (i.e. the origin) and will have a value of -2a. Thus, the local shape of a curve could be defined as a parabola using this constraint. Figure 4.2 shows a set of parabolas with varying values of a. Finally, the resulting equations and maximum curvatures for this family of parabolas are

[3] An analogous curve $x - ay^2 = 0$ could also be defined with the same analysis.

shown in Table 4.1. As discussed above, the maximum curvature for each of these curves is located at the origin.

$$\kappa(x,y)=\frac{-2a}{\left(1+4a^2x^2\right)^{3/2}} \qquad 4.5$$

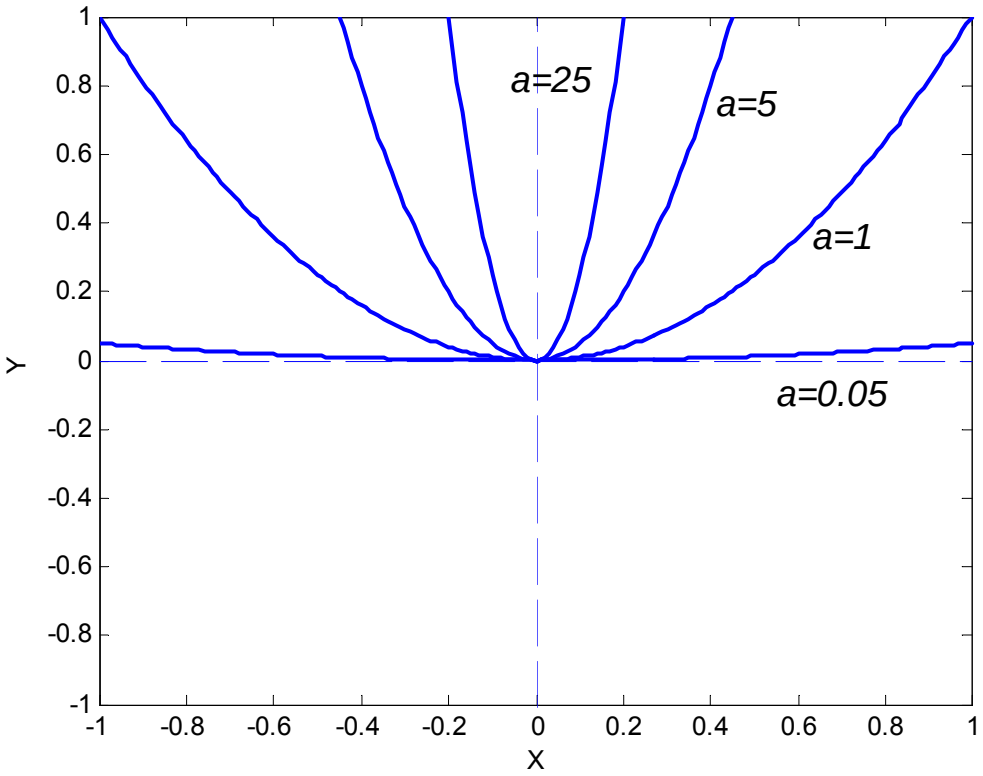


Figure 4.2. Family of Parabolas

a	**Implicit Equation**	**Parametric Equation**	$\lvert\kappa_{\max}\rvert$ **(**κ **at origin)**
0.05	$y-0.05x^2$	$x(u)=u$ $y(u)=0.05u^2$	0.1
1	$y-x^2$	$x(u)=u$ $y(u)=u^2$	2
5	$y-5x^2$	$x(u)=u$ $y(u)=5u^2$	10
25	$y-25x^2$	$x(u)=u$ $y(u)=25u^2$	50

Table 4.1. Curve Parameters for Family of Parabolas

4.3.2. Circle

A circle of radius r centered at the origin is represented by the implicit equation shown in Equation 4.6. The curvature can once again be calculated by finding the partial derivatives and substituting them into Equation 4.4: $f_x = 2x$, $f_y = 2y$, $f_{xx} = f_{yy} = 2$, $f_{xy} = 0$. Once these values are substituted, the result is a constant value of $\kappa = 1/r$. This result is expected, because the radius of curvature should be constant along a circle. Figure 4.3 further shows the relationship between radius and curvature. As the curvature of the curves passing through the origin increases, the radius of the circles becomes smaller. This represents a sharper bend in the curve.

$$x^2 + y^2 - r^2 = 0 \qquad \textbf{4.6}$$

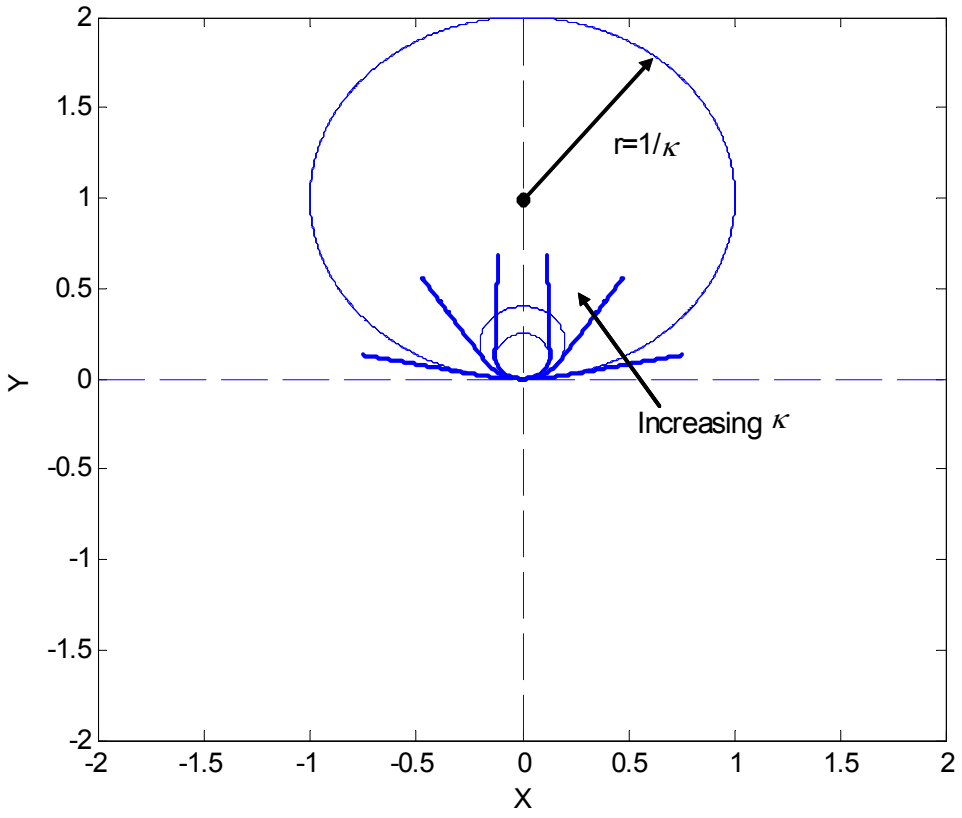


Figure 4.3. Effects of Varying Curvature

The simplest way to write a parametric description of a circle is to use sine and cosine functions as shown in Equation 4.7. However, this is not technically an algebraic (i.e. polynomial) description. A rational parameterization can be derived as shown in Equation 4.8. Note that this description requires the independent parameter u to go infinite to fully trace the circle.

$$\left.\begin{aligned} x(u) &= r\cos(u) \\ y(u) &= r\sin(u) \end{aligned}\right\}, \; u \in [-2\pi, 2\pi] \tag{4.7}$$

$$\left.\begin{aligned} x(u) &= \frac{r(1-u^2)}{1+u^2} \\ y(u) &= \frac{2ur}{1+u^2} \end{aligned}\right\}, \; u \in [-\infty, \infty] \tag{4.8}$$

4.3.3. Ellipse

An ellipse can be thought of as a more general description of a circle. The general equation for an ellipse centered at the origin is given in Equation 4.9. In this equation, a and b describe the focal lengths of the ellipse as shown in Figure 4.4. If $a>b$, the major axis will be the x-axis. If $a<b$, the major axis will be the y-axis. In the case that $a=b$, this simplifies to the equation of a circle.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0 \tag{4.9}$$

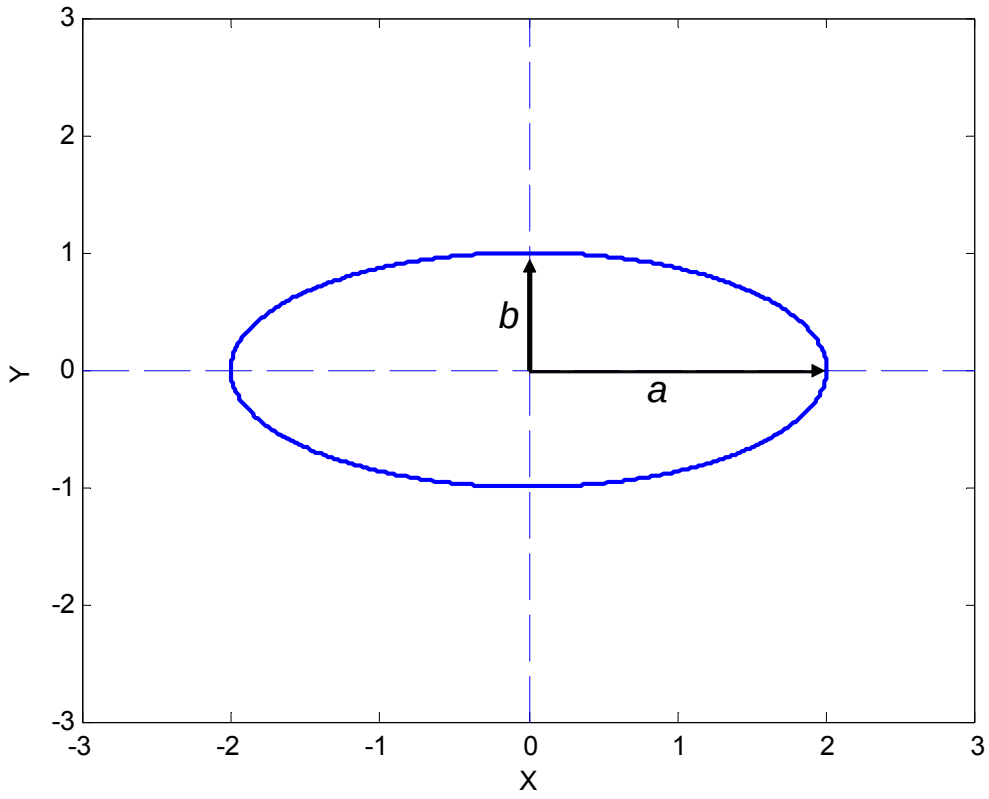


Figure 4.4. Ellipse Centered at the Origin

As before, the curvature along the ellipse can be calculated by substituting the partial derivatives back into implicit equation for curvature. The resulting curvature equation is shown in Equation 4.10 where $A = 1/a^2$ and $B = 1/b^2$. Once again, this closed-form solution ends up being very complex. However, it still provides the ability to define the local shape of an ellipse at any point in terms of curvature.

$$\kappa(x,y) = \frac{8AB}{\left(4A^2x^2 + 4B^2y^2\right)^{3/2}} \qquad \textbf{4.10}$$

Finally, the parametric forms of the equation can be formulated. Equation 4.11 shows a simple formulation using sine and cosine functions. Equation 4.12 shows an algebraic parameterization.

$$x(u) = a\cos(u)$$
$$y(u) = b\sin(u) \qquad \textbf{4.11}$$

$$x(u) = \frac{-\left(\frac{2}{b}\right)u}{\frac{1}{a^2}+\left(\frac{1}{b^2}\right)u^2},\ y(u) = \frac{-\left(\frac{1}{b}\right)u^2+\left(\frac{b}{a^2}\right)}{\frac{1}{a^2}+\left(\frac{1}{b^2}\right)u^2} \qquad \textbf{4.12}$$

4.3.4. Cusp

As mentioned in Section 2.1.2.2, a cusp is a singular point where two branches of a curve meet with a shared tangent. Another condition at this point is the determinant of the Hessian matrix will be zero (as described in Section 2.1.2.2). Because the cusp is also a singularity (or critical point), this yields three conditions for the existence of a cusp: (i) $f(x,y)=0$, (ii) $\frac{\partial f}{\partial x}=\frac{\partial f}{\partial y}=0$, (iii) $\frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2}-\left(\frac{\partial^2 f}{\partial x \partial y}\right)=0$. The cusp explored in Section 2.1.2.2 can be rewritten as $ay^2-x^3=0$ with the parameter *a* used to generate a family of curves. This family of curves is shown in Figure 4.5.

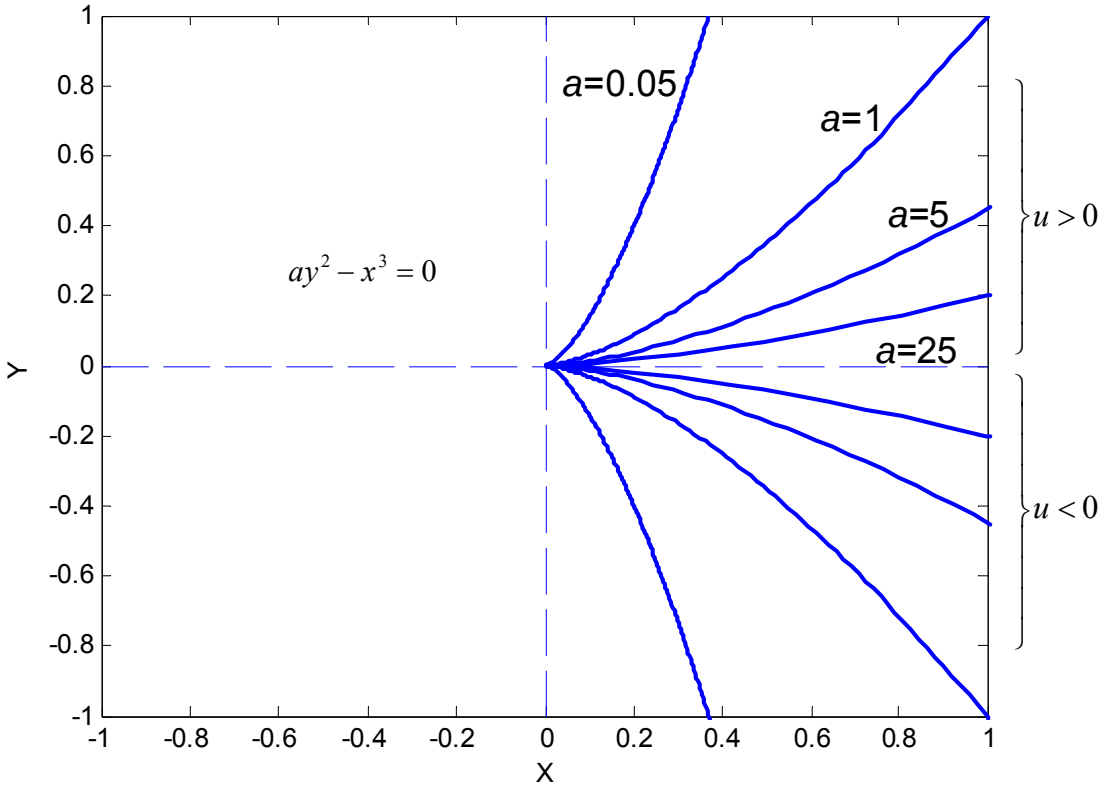


Figure 4.5. Family of Cusps

Using the same procedure as before, the curvature of this curve can be calculated as shown in Equation 4.13. This equation shows that the curvature would go to infinity at the singular point (0,0). Another useful way to look at a cusp is to examine its tangent vector. For an implicit curve, the tangent vector is given as $\left[-f_y, f_x\right]$ or $\left[-2ay, 3x^2\right]$ for this curve. From this equation for the tangent vector, the *x* component of the tangent will flip directions when the *y* axis is crossed.

$$\kappa(x,y) = \frac{18ax^4 - 24a^2xy^2}{\left(9x^4 + 4a^2y^2\right)^{3/2}} \qquad \textbf{4.13}$$

The parametric form of this curve can easily be derived as shown in Equation 4.14. The tangent vector of the parametric form can be formulated

as $\left[\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}\right] = \left[2au, 3au^2\right]$. From this equation for the tangent vector, it can be seen that the *x* component again flips directions when the *u* parameter crosses zero. Thus, the two parameter ranges $u \in [0, \infty]$ and $u \in [0, -\infty]$ trace the two branches of the curve. Table 4.2 summarizes the various curve parameters for the family of cusps described in this section.

$$\left.\begin{aligned} x(u) &= au^2 \\ y(u) &= au^3 \end{aligned}\right\}, u \in [-\infty, \infty] \qquad \textbf{4.14}$$

a	**Implicit Equation**	**Parametric Equation**	$\lvert\kappa_{\max}\rvert$ **(*κ* at origin)**
0.05	$0.05y^2 - x^3 = 0$	$x(u) = 0.05u^2$ $y(u) = 0.05u^3$	∞
1	$y^2 - x^3 = 0$	$x(u) = u^2$ $y(u) = u^3$	∞
5	$5y^2 - x^3 = 0$	$x(u) = 5u^2$ $y(u) = 5u^3$	∞
25	$25y^2 - x^3 = 0$	$x(u) = 25u^2$ $y(u) = 25u^3$	∞

Table 4.2. Curve Parameters for Family of Cusps

To further illustrate the behavior of curvature at a cusp point, Figure 4.6 shows the curvature profiles for the different values of *a* across the parameter range $u \in [-1, 1]$. This plot shows that the curvature values approach infinity as the parameter *u* approaches zero. However, it is hard to define a clear physical meaning to an infinite curvature. Thus, it is hard to define a simple constraint based on curvature that defines the shape of a cusp.

A more geometrically intuitive way to think of this kind of cusp is as a point where the unit tangent vector switches direction.

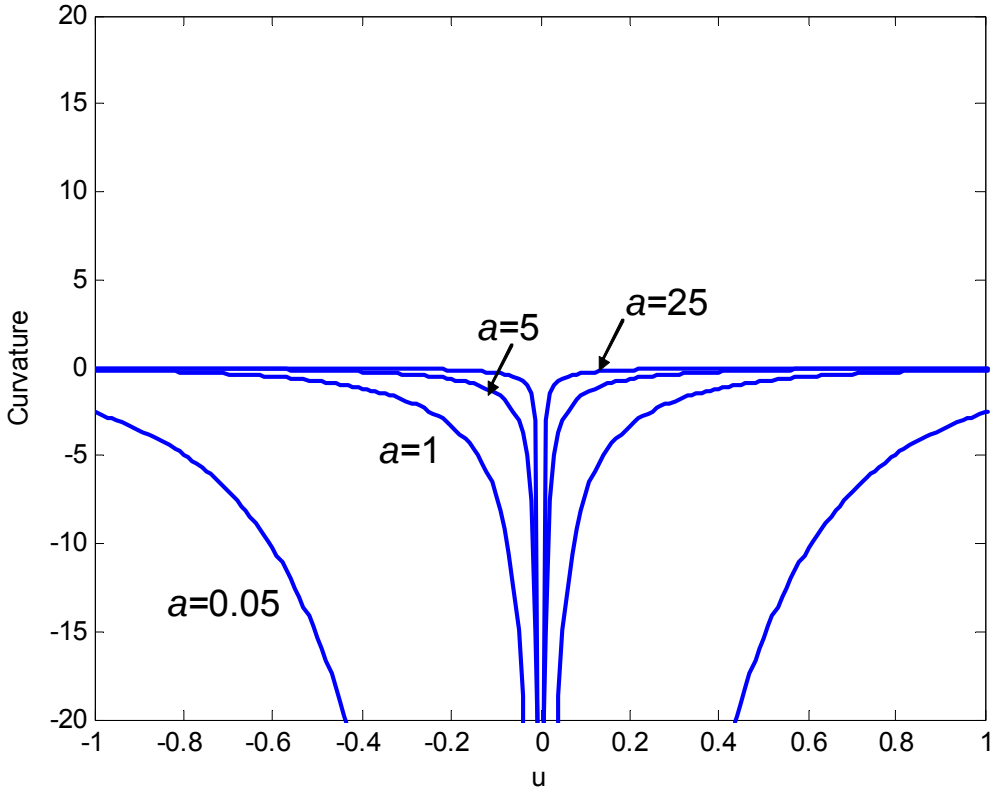


Figure 4.6. Curvature Profiles for Family of Cusps

Section 3.4.4 introduced a method for capturing implicit forms of curves in barycentric A-Spline representations. To deal with singular points such as cusps, this method will simply break the curve into two branches at the singular point. This allows for the curve to be rendered/traced without dealing with the mathematics at the singular point. For this case, the coefficients can be calculated in terms of the cusp parameter a as $b_{003} = -1$, $b_{012} = -1$, $b_{021} = -1 + \frac{a}{3}$, $b_{030} = -1 + a$, $b_{120} = a/3$ and all other coefficients zero. While this formulation is not a focus of this research, it remains an interesting and potentially useful method for generating complex planar curves for path planning. Figure 4.7 shows the cusp being generated with the use of A-Splines.

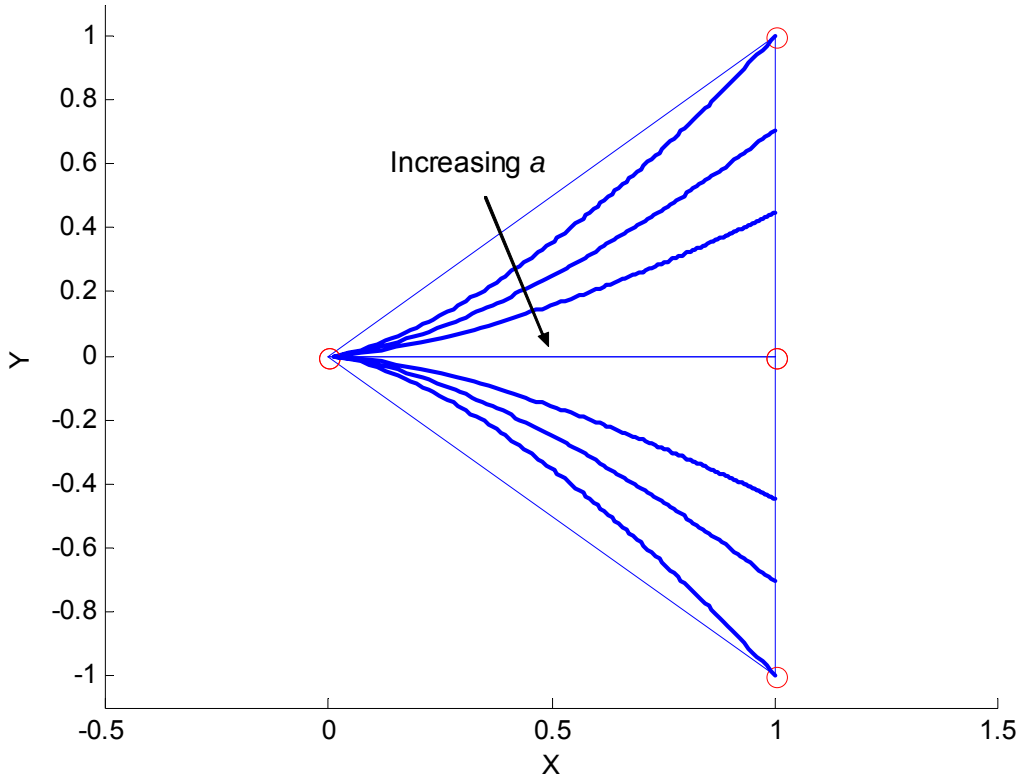


Figure 4.7. Family of Cusps in A-Spline Representation

4.3.5. Inflection Point

Unlike the previous examples that described geometric shapes, an inflection point is a completely local phenomenon. In planar curves, an inflection point occurs whenever the sign of the curvature changes. Thus, the condition $\kappa = 0$ is necessary but not sufficient. Figure 4.8 illustrates this point. All three of these curves have $\kappa = 0$ at the origin, but only the middle one has an inflection point while the other two curves just have local minima and maxima.

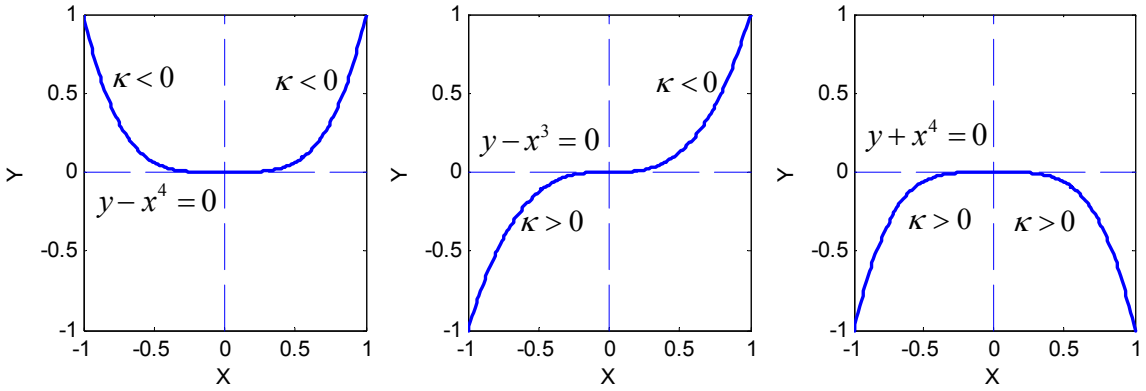


Figure 4.8. Curves with Zero Curvature

Another way to look at an inflection point is to observe the tangent and normal vectors along a curve. At an inflection point, the normal vector will flip directions while the tangent vector will not. Figure 4.9 shows a plot of the curve $y - x^3 = 0$ along with its tangent and normal vectors. This shows the normal vector switching directions as it crosses the origin and represents the curve beginning to bend in the opposite direction. This curve can also be represented parametrically as $x(u) = u, y(u) = u^3$. Table 4.3 shows the values of the tangent and normal vectors for several values of u along the curve. This table shows numerically that the normal vector switches directions (i.e. inverts) as u crosses from negative to positive.

u	x	y	$\hat{\mathbf{T}}$ [4]	$\hat{\mathbf{N}}$ [4]
-0.5	-0.5	-0.125	[0.8 0.6]	[0.6 -0.8]
-0.1	-0.1	-0.001	[0.9996 0.03]	[0.03 -0.9996]
0.1	0.1	0.001	[0.9996 0.03]	[-0.03 0.9996]
0.5	0.5	0.125	[0.8 0.6]	[-0.6 0.8]

Table 4.3. Curve Parameters around an Inflection Point

[4] These values represent the x and y directions for the tangent and normal vectors.

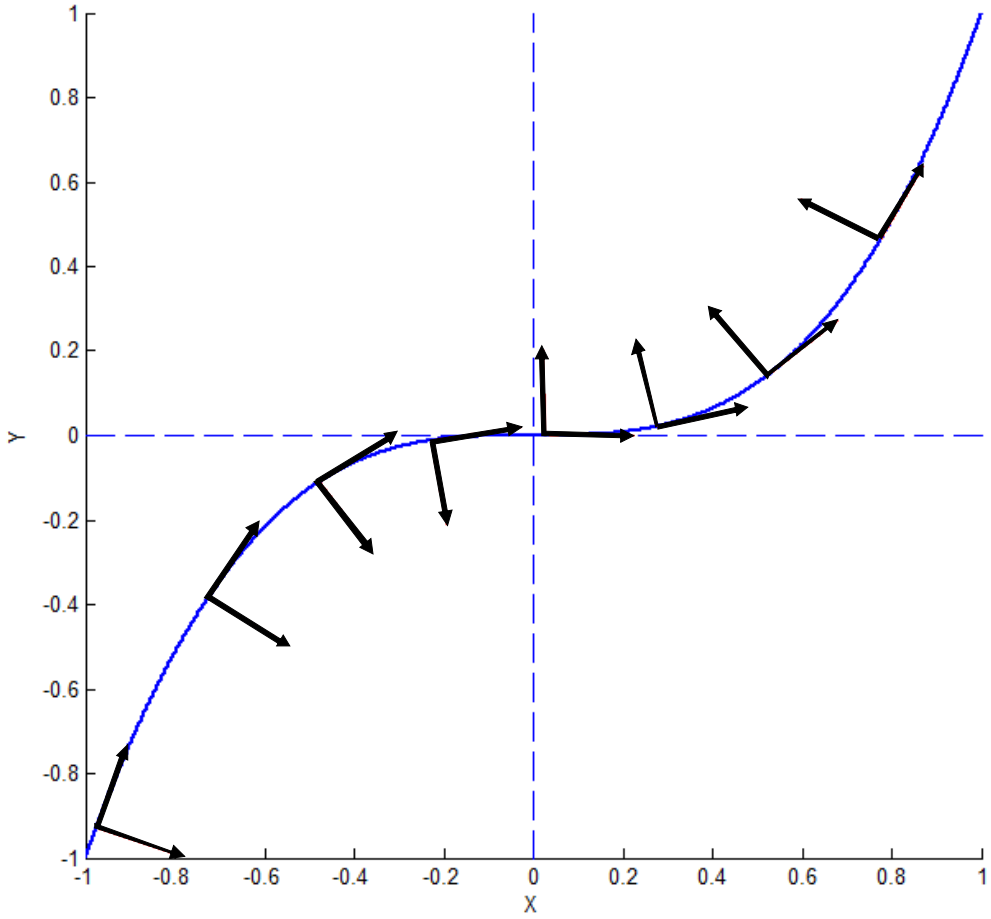


Figure 4.9. Planar Curve with an Inflection Point

4.4. SPATIAL GEOMETRIC SHAPES

This section will begin to explore more complex spatial geometries. When moving to the spatial domain, it becomes more difficult to deal directly with implicit and parametric forms of curves. Thus, a different approach is presented in this section. This approach involves studying the effects of curvature and torsion on the local geometry of a curve by generating curves based on their curvature and torsion values. Curvature and torsion are high-order properties of curves as shown in Equations 4.15 and 4.16 and fully define the local geometry of a curve. Once these properties are well understood

physically, the parametric constraints shown on the right side of Equations 4.15 and 4.16 can be defined. Then, these parametric constraints can be blended together to form an overall path plan.

$$\kappa = f\left(\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \frac{d^2x}{du^2}, \frac{d^2y}{du^2}, \frac{d^2z}{du^2}\right) \quad \textbf{4.15}$$

$$\tau = f\left(\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \frac{d^2x}{du^2}, \frac{d^2y}{du^2}, \frac{d^2z}{du^2}, \frac{d^3x}{du^3}, \frac{d^3y}{du^3}, \frac{d^3z}{du^3}\right) \quad \textbf{4.16}$$

Before specific geometric shapes are explored, a brief description of how to define curves in terms of curvature and torsion is presented (first introduced in Section 2.4). These curves are defined locally relative to a fixed Frenet Frame. For the purposes of this research, this frame is placed at the origin with the tangent, normal, and bi-normal vectors lined up with the *x*, *y*, and *z* axes respectively. However, this frame could be placed at any location and orientation, and the geometry of the curve relative to the frame will remain the same. Figure 4.10 shows the effect of various curvature values with zero torsion. Because the torsion is zero, the curve remains in the osculating plane (*xy* plane in this example), and the curvature values affect how sharply the curve bends around the frame of interest.

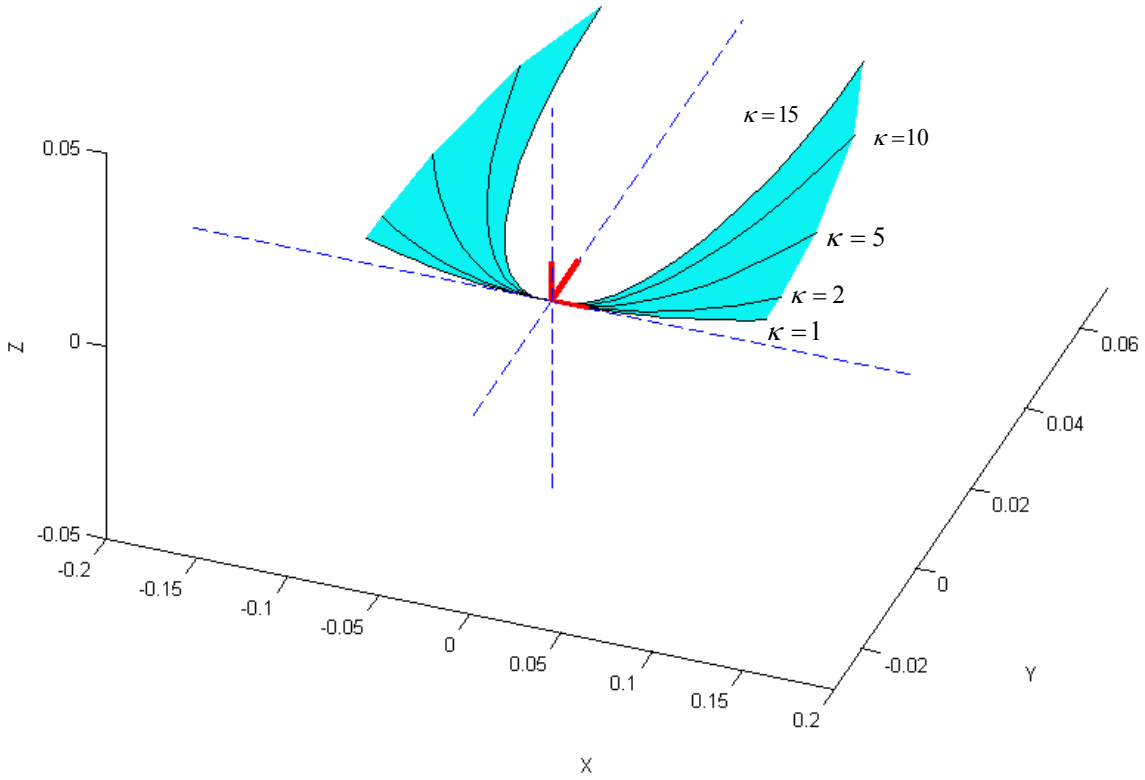


Figure 4.10. Local Effects of Curvature

Figure 4.11 shows a curve with varying torsion and a constant curvature[5] ($\kappa = 1$). There a few things to note from this figure. First, torsion has an effect on the movement of the curve in the bi-normal direction. Thus, because the torsion is positive when both approaching and leaving the local frame, the curve moves in the positive z direction. Also, torsion is a signed value (unlike curvature which is always defined to be positive), and a negative torsion will result in movement in the opposite direction of the bi-normal as in Figure 4.12. Finally, the scale of the changes in the z direction is much smaller than changes in the osculating plane. Thus, numerically, torsion has a smaller effect on the shape of the curve than an equivalent numeric value of curvature.

[5] When curvature is zero, torsion is undefined. Thus, whenever a nonzero value of torsion is provided, a non-zero value of curvature is also needed.

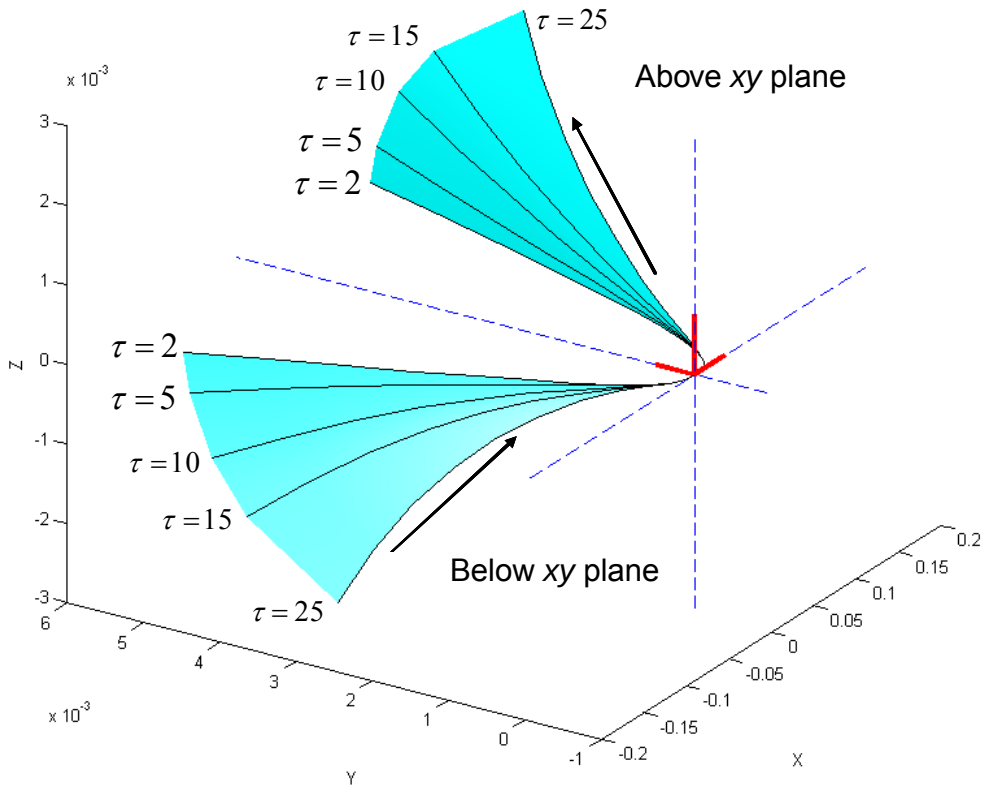


Figure 4.11. Local Effects of Torsion

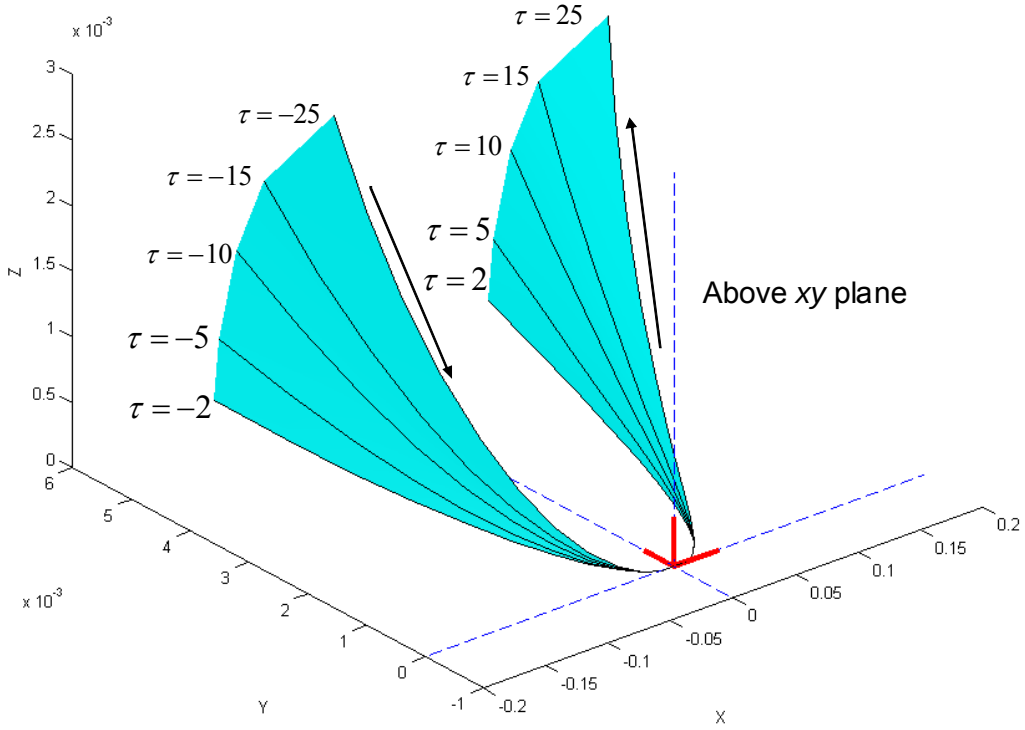


Figure 4.12. Positive vs. Negative Torsion

Another important aspect of the local effects of curvature and torsion is the coupling between them. In terms of the motion relative to the osculating plane, the effect of curvature dominates, and torsion has smaller influence. This can be observed locally in Figure 4.11 and Figure 4.12 where the varying torsion has an effect mainly on the movement in the z direction. Thus, holding a constant curvature and varying the torsion will only change how fast the curve is moving in the bi-normal direction. However, holding torsion constant and varying the curvature has a much different effect as shown in Figure 4.13. Thus, torsion mainly affects the motion relative to the bi-normal direction, while curvature has a strong influence on the motion of the curve in all directions.

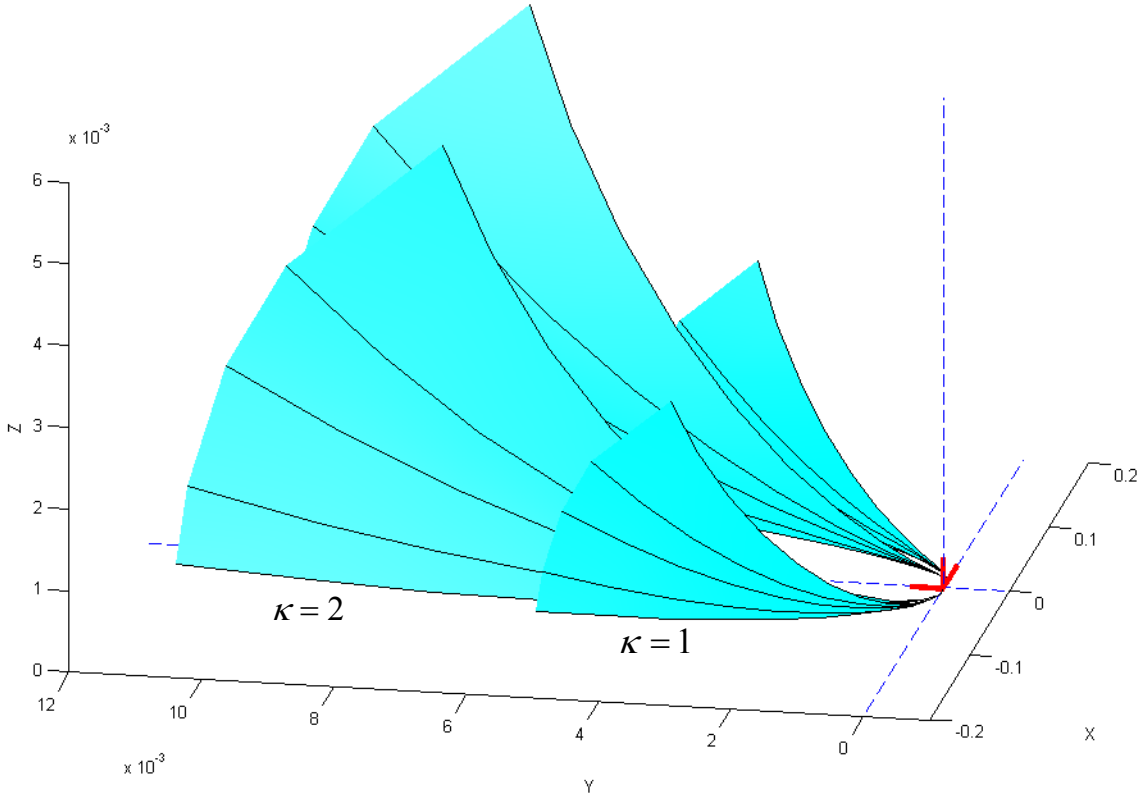


Figure 4.13. Varying Curvature with Constant Torsion

4.4.1. Helix

A helix is a spatial geometry with a constant radius and pitch. The curve can be thought of running along the surface of a cylinder with the helix radius being the cylinder radius and the pitch being the distance travelled along the cylinder's axis for each full revolution. An example of this is shown in Figure 4.14. One simple parametric representation for a helix is shown in Equation 4.17.

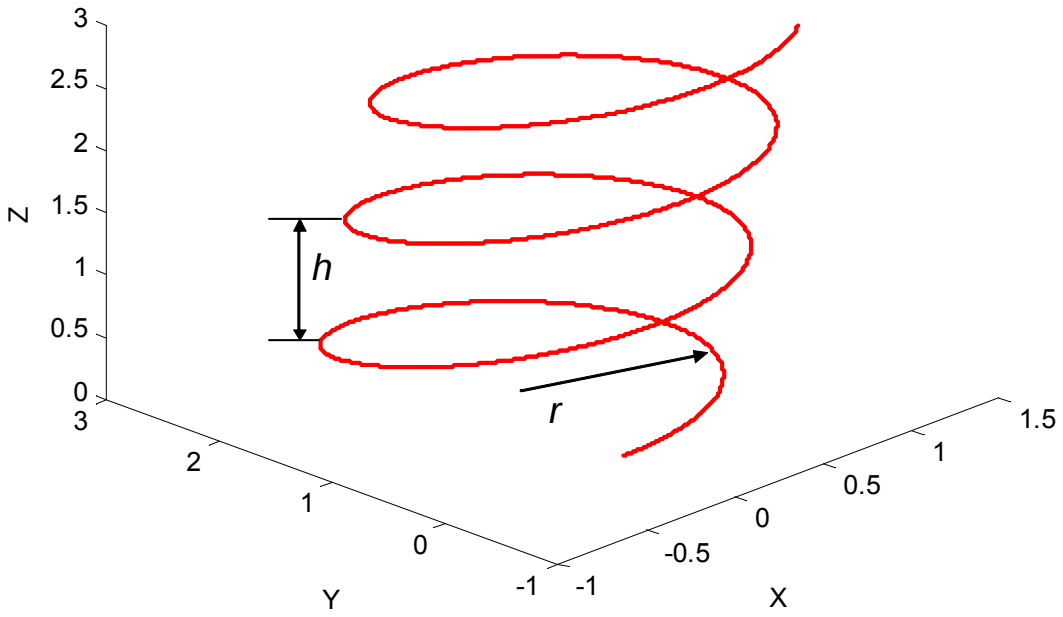


Figure 4.14. Helical Curve with Defined Radius and Pitch

$$\begin{aligned} x(u) &= r\cos(u) \\ y(u) &= r\sin(u) \\ z(u) &= \frac{h}{2\pi}u \end{aligned} \qquad \textbf{4.17}$$

The geometry of a helix can also be described in terms of the curve's curvature and torsion. For a helix, the curvature and torsion will both be constant and non-zero. These relationships are shown in Equation 4.18 and Equation 4.19 [26] where $l = h/2\pi$.

Thus, the desired radius and pitch for a helical motion can be entirely defined in terms of curvature and torsion constraints.

$$\kappa = \frac{r}{r^2 + l^2} \quad \textbf{4.18}$$

$$\tau = \frac{l}{r^2 + l^2} \quad \textbf{4.19}$$

The inverse relationships can also be easily derived and are shown in Equation 4.20 and 4.21.

$$r = \frac{\kappa}{\kappa^2 + \tau^2} \quad \textbf{4.20}$$

$$l = \frac{\tau}{\kappa^2 + \tau^2} \quad \textbf{4.21}$$

As mentioned before, a helix can be thought of as a curve running along the surface of a cylinder, and this interpretation is a good way of visualizing the effects of curvature and torsion on a shape of a helix. Figure 4.15 shows three helices with varying values for curvature running along the surfaces of cylinders. Table 4.4 shows the relevant curve parameters for these helices as well as an equivalent parametric representation. This data shows that as curvature increases both the radius and pitch of the helix decreases. However, the pitch decreases at a faster rate than the radius. This is consistent with both Equations 4.20 and 4.21 as well as the curves in Figure 4.15.

κ	τ	$\kappa^2+\tau^2$	r	l	Parametric Equation
1	1	2	0.5	0.5	$x(u)=0.5\cos(u)$ $y(u)=0.5\sin(u)$ $z(u)=0.5u$
2	1	5	0.4	0.2	$x(u)=0.4\cos(u)$ $y(u)=0.4\sin(u)$ $z(u)=0.2u$
5	1	26	0.1923	0.0385	$x(u)=0.1923\cos(u)$ $y(u)=0.1923\sin(u)$ $z(u)=0.0385u$

Table 4.4. Curve Parameters for Helices of Varying Curvature

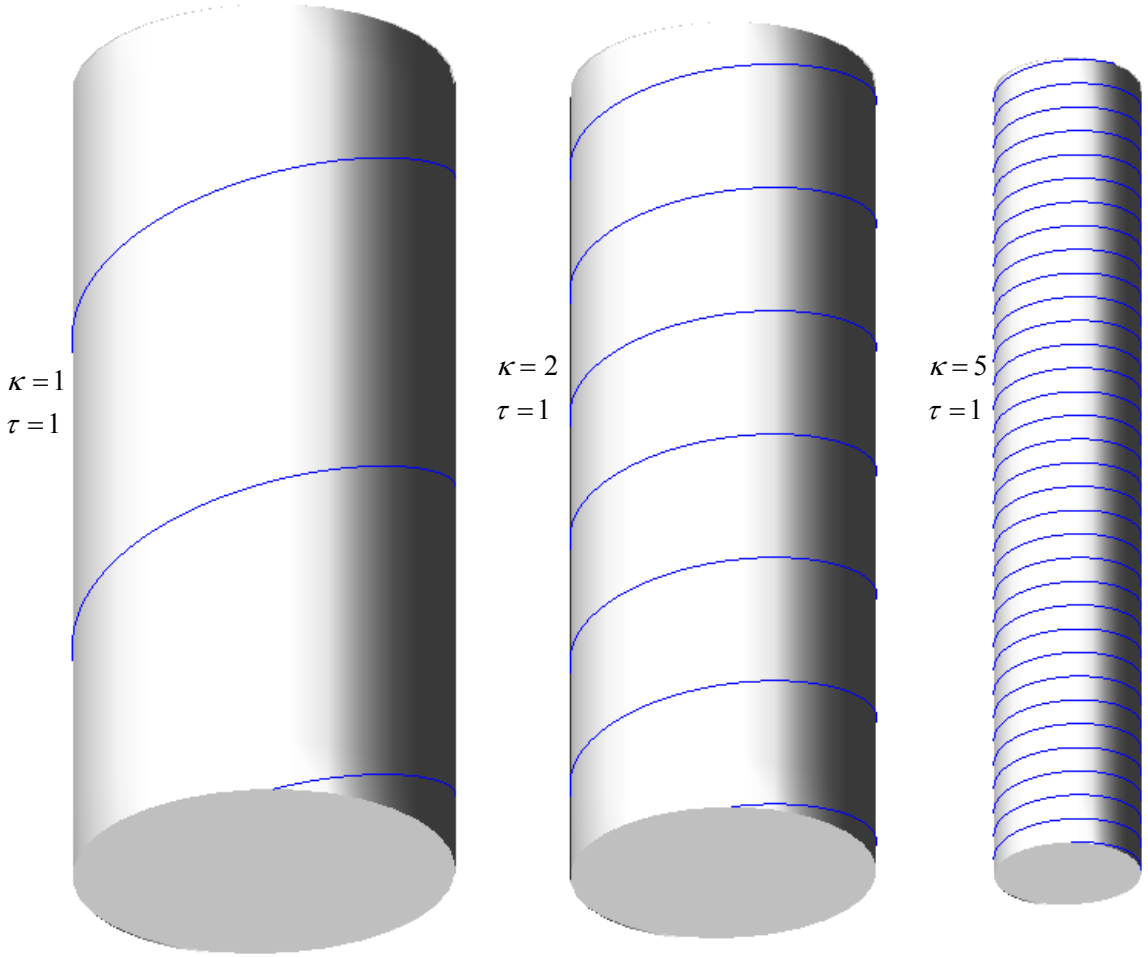


Figure 4.15. Effect of Varying Curvature on a Helix

Now, the effect of varying the torsion of the helix can be visualized as in Figure 4.16. The relevant curve parameters are tabulated in Table 4.5. The results are similar to the results of varying curvature except this time the radius decreases at a faster rate than the pitch. The result is a curve that twists at faster rate. For extremely high values of torsion, this would approximate a straight line with the Frenet Frame rotating around it. Now, using the results of this section, a helical curve can be defined by its radius and its pitch which provides a good physical understanding. Then, Equation 4.18 and 4.19 can be used to convert the radius and pitch into constraints on the curvature and torsion.

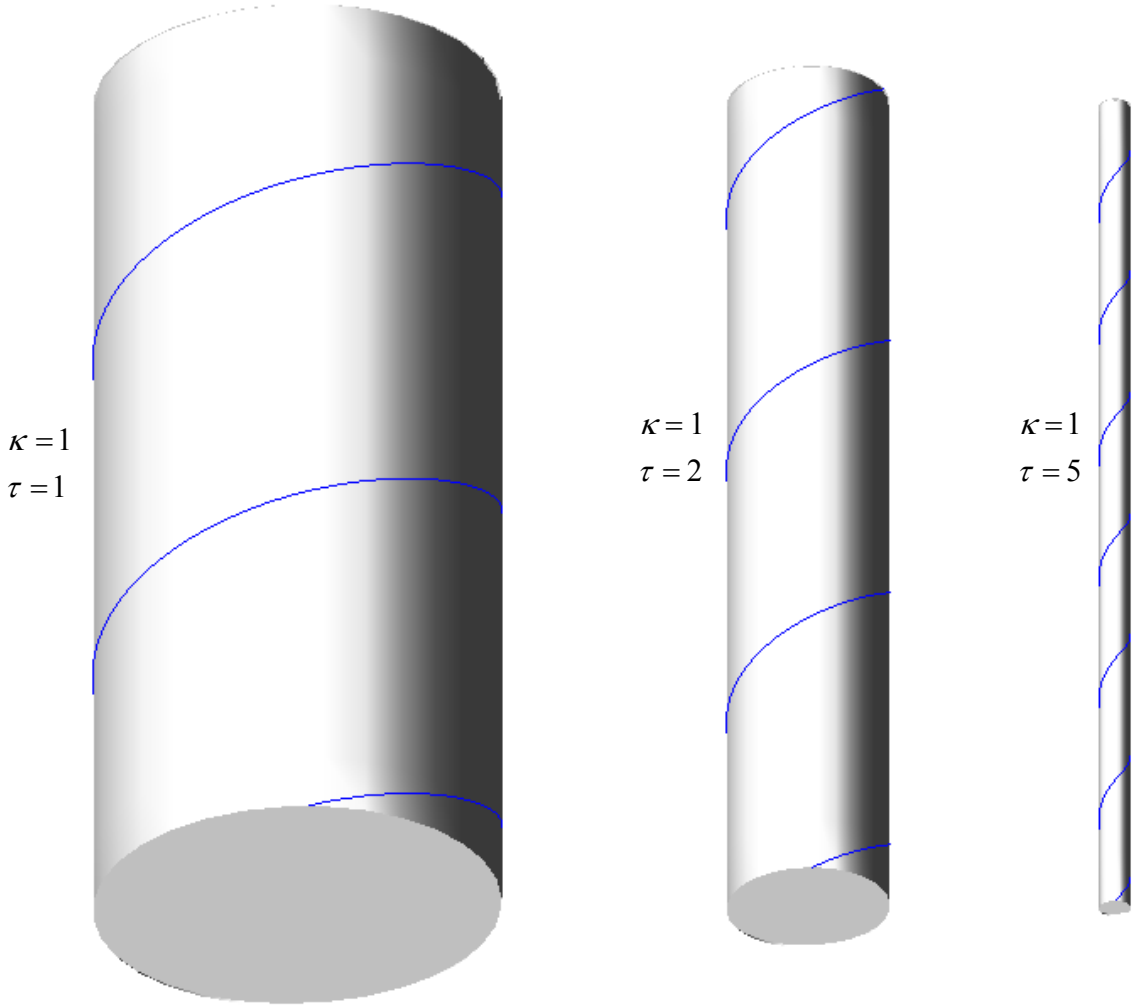


Figure 4.16. Effect of Varying Torsion on a Helix

κ	τ	$\kappa^2+\tau^2$	r	l	Parametric Equation
1	1	2	0.5	0.5	$x(u)=0.5\cos(u)$ $y(u)=0.5\sin(u)$ $z(u)=0.5u$
1	2	5	0.2	0.4	$x(u)=0.2\cos(u)$ $y(u)=0.2\sin(u)$ $z(u)=0.4u$
1	5	26	0.0385	0.1923	$x(u)=0.0385\cos(u)$ $y(u)=0.0385\sin(u)$ $z(u)=0.1923u$

Table 4.5. Curve Parameters for Helices of Varying Torsion

4.4.2. Spatial Cusp

Geometrically, a cusp in a spatial curve is similar to a cusp in a planar curve. However, it is much more difficult to define mathematically, because the curve cannot be defined in implicit form. In a spatial curve with a polynomial or rational parameterization, a necessary condition for a cusp is that the first derivative vector is undefined [30][31]. This results in a discontinuity in the unit tangent vector. Consider the curve defined by Equation 4.22. The first derivative vector can be calculated as $\left[15u^4, 20u^3, 5u^4\right]$, and it can be seen that this vector vanishes (i.e. [0,0,0]) at the point u=0. Further, the first derivative vector shows that the y direction of the tangent will flip as u passes from positive to negative due to the u^3 term. Figure 4.17 shows a plot of this cusp.

$$\begin{aligned} x(u) &= 3u^5 \\ y(u) &= 5u^4 \\ z(u) &= u^5 \end{aligned} \qquad \textbf{4.22}$$

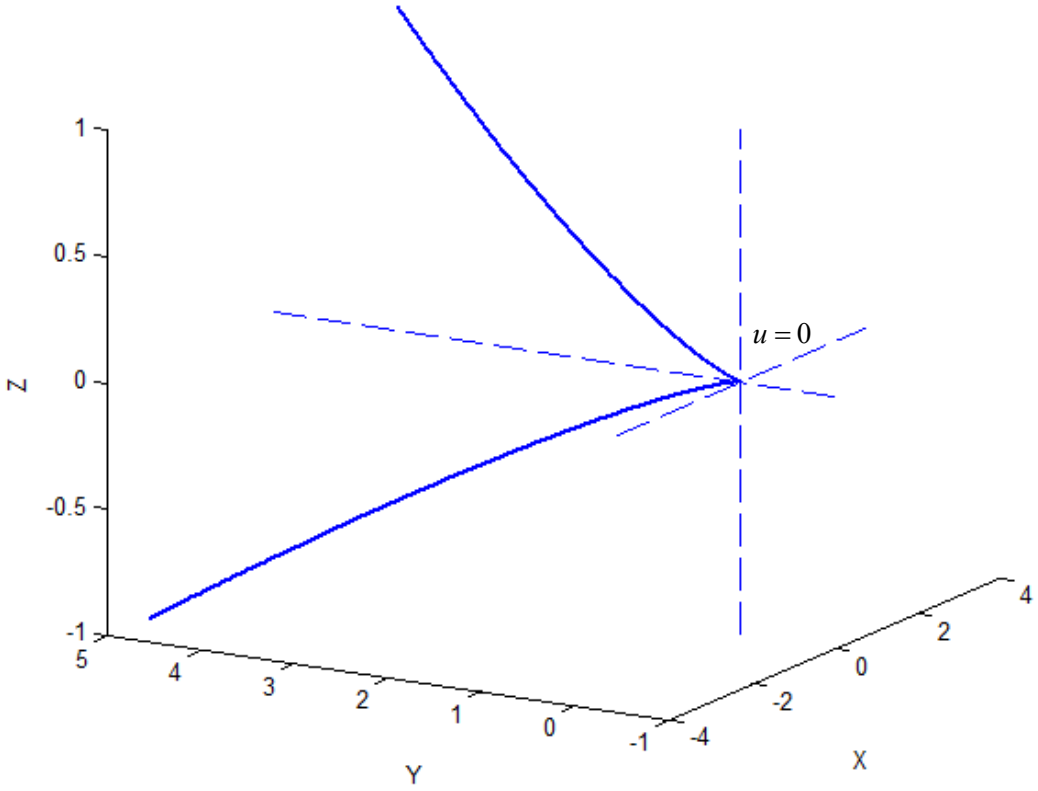


Figure 4.17. Spatial Cusp

However, the main geometric significance of a cusp for the purpose of path planning is that the tangent vector switches direction. Thus, this information can be used to generate a cusp at a specific point, and the higher-order properties of curvature and torsion can be used to generate the shape of the curve approaching and leaving the cusp area. This leads to a more intuitive geometric understanding of a cusp. Figure 4.18 shows a simple cusp with varying curvature values, and a torsion of zero. In this example, the curve remains in the osculating plane, and the increasing values of curvature define how sharply the curve bends around the tangent vector.

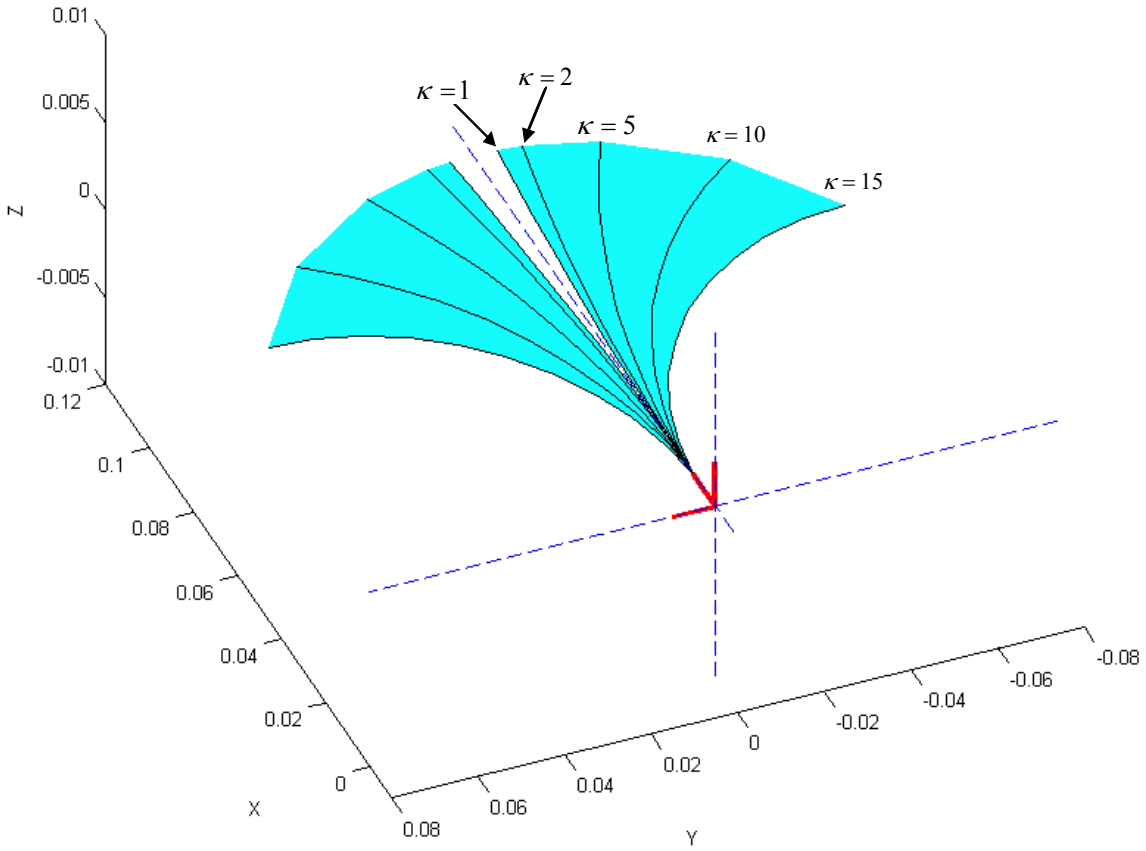


Figure 4.18. Planar Cusp Defined with Varying Curvatures

Figure 4.19 shows the effect of varying curvatures with non-zero torsion. This shows that the increasing value of curvature affects both the movement in the osculating plane as well as how quickly the curve twists out of this plane in the bi-normal direction. It should be noted that the rest of this chapter will concentrate on the effects of curvature and torsion on the local geometry of curves. It will be shown how to use these properties to develop parametric constraints for curves in the next chapter.

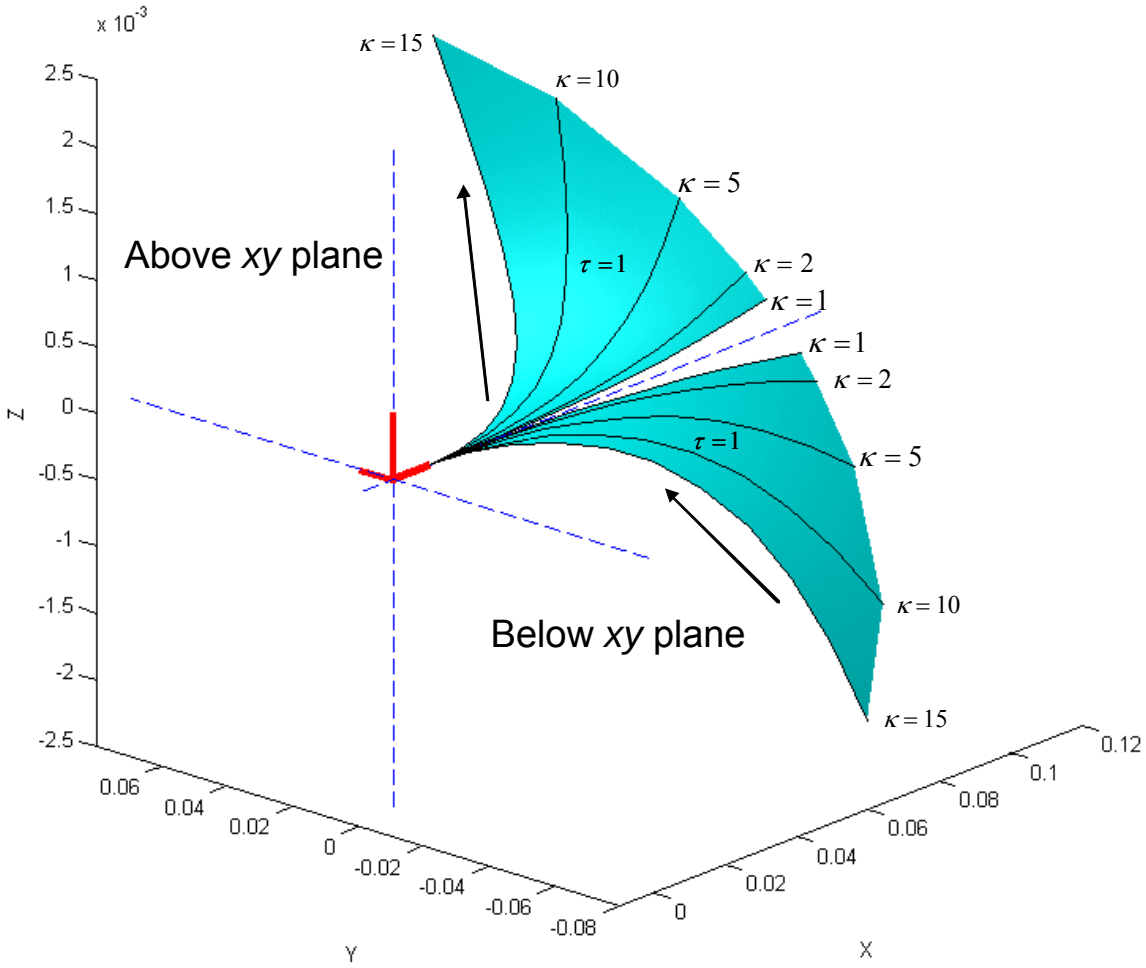


Figure 4.19. Cusp with Constant Torsion and Varying Curvature

Figure 4.20 shows the effect of a constant curvature with a varying torsion. As this plot once again shows, higher values of torsion result in faster motions out of the osculating plane and in the direction of the bi-normal. Figure 4.21 shows a similar plot that uses negative torsion. Thus, a cusp can be defined geometrically as a point where the tangent vector inverts. Then, constraints based on curvature and torsion can be used to define the local geometry of the curve around the cusp.

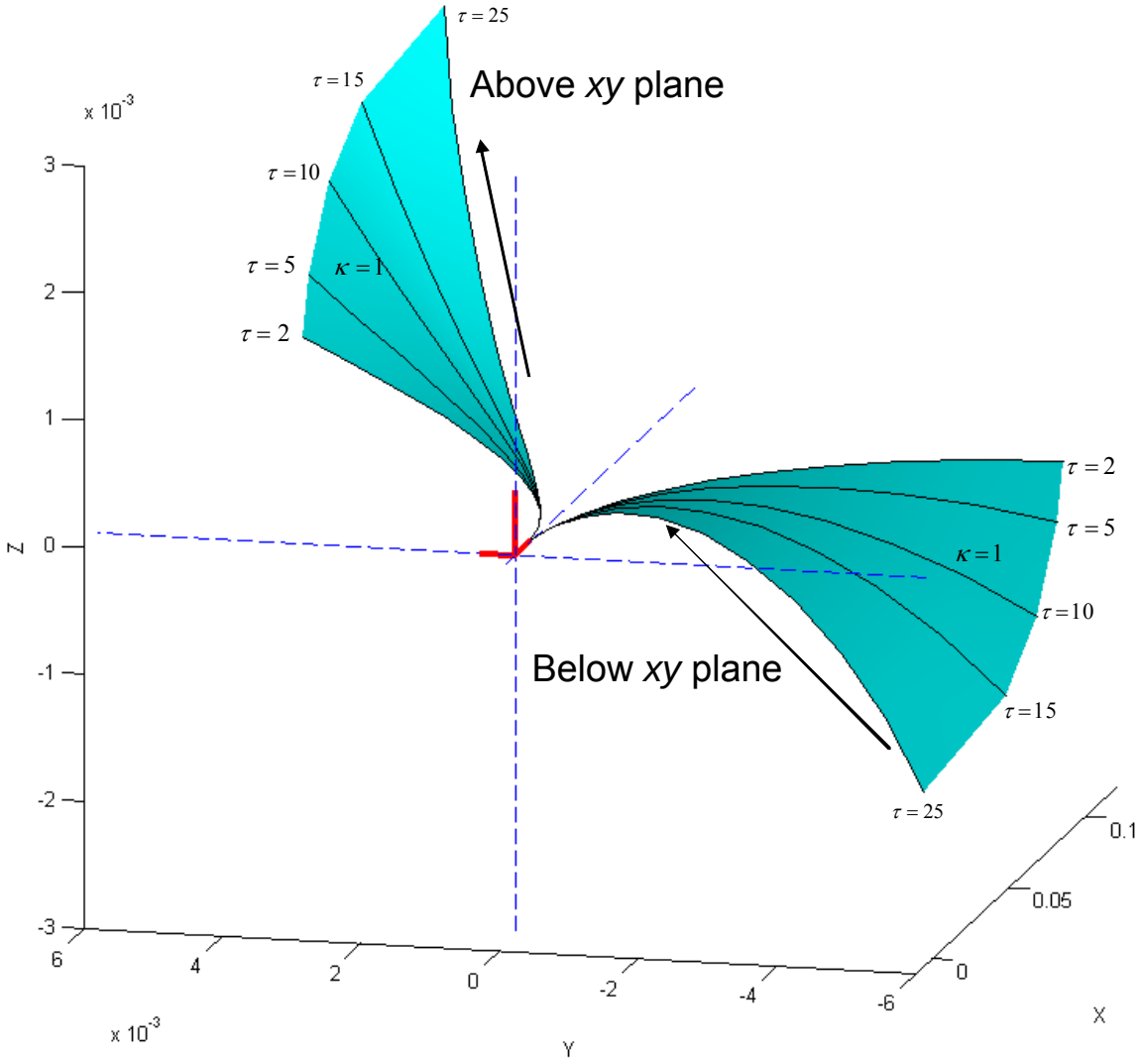


Figure 4.20. Spatial Cusp with τ >0 Approaching and τ >0 Leaving

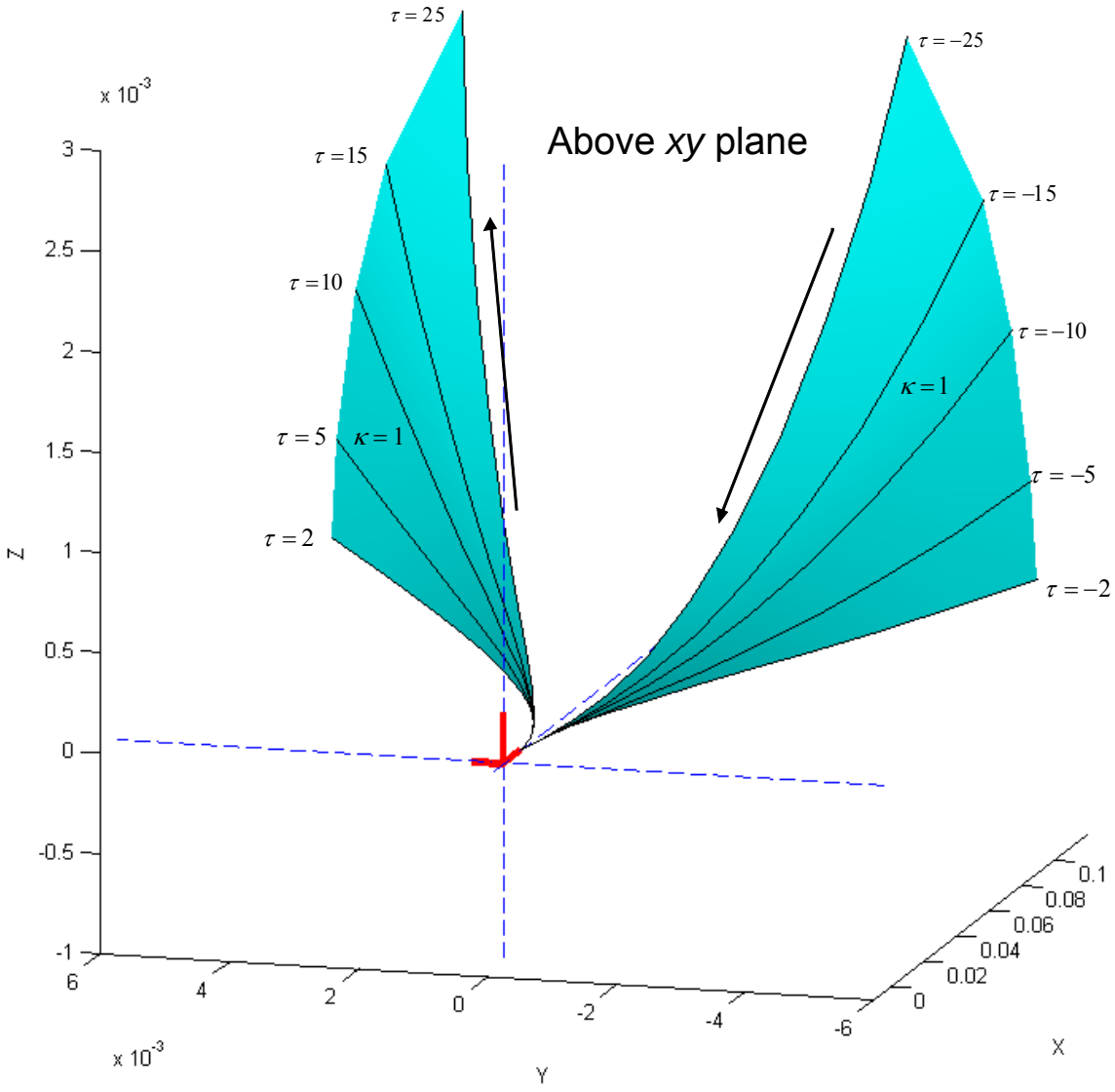


Figure 4.21. Spatial Cusp with τ <0 Approaching and τ >0 Leaving

4.4.3. Spatial Saddle Point

A spatial saddle point occurs when the torsion of the curve is zero. This represents a point where the shape of the curve is instantaneously planar. Non-zero higher-order derivatives of torsion ensure that the curve will not remain in the plane and also dictate

the shape of the curve in the region approaching and leaving the saddle point. Figure 4.22 shows an example of one type of spatial saddle where $\tau' = \frac{d\tau}{ds}$.

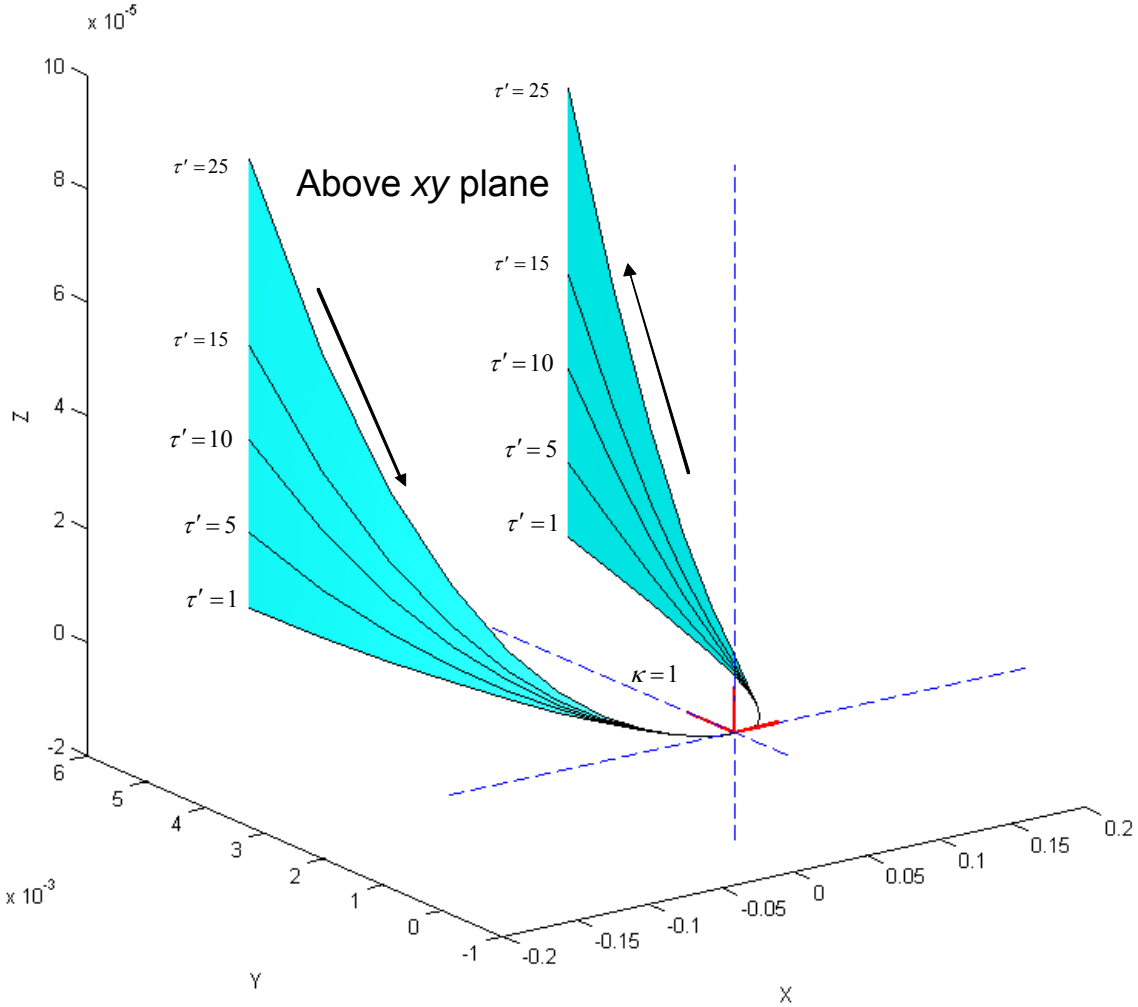


Figure 4.22. Saddle Point 1

Figure 4.23 shows the torsion profile around the point of interest. The center of this graph represents the torsion at the defined frame (i.e. the saddle point). Then, the values on the left represent the torsion as the curve is approaching the frame, and the values on the right represent the torsion as the curve is leaving the frame. This plot shows that the torsion is negative with a positive τ' during the approach, and the torsion crosses zero and becomes positive on the leaving side.

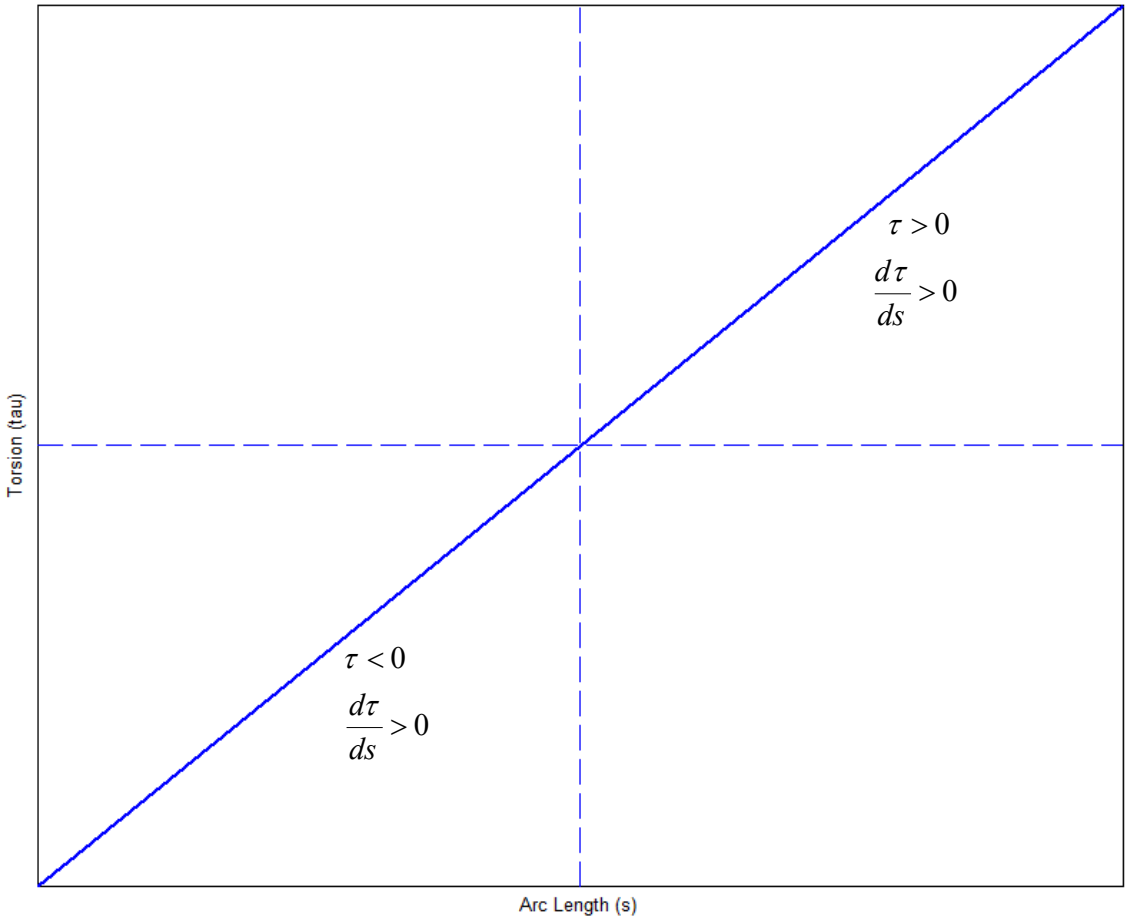


Figure 4.23. Torsion Profile for Saddle Point

A similar type of saddle point is shown in Figure 4.23 that shows the effect of a negative τ'. The torsion profile for this saddle is shown in Figure 4.24. This shows the torsion will be positive both approaching and leaving the frame. Thus, the curve moves in the positive *z* direction into and out of the plane.

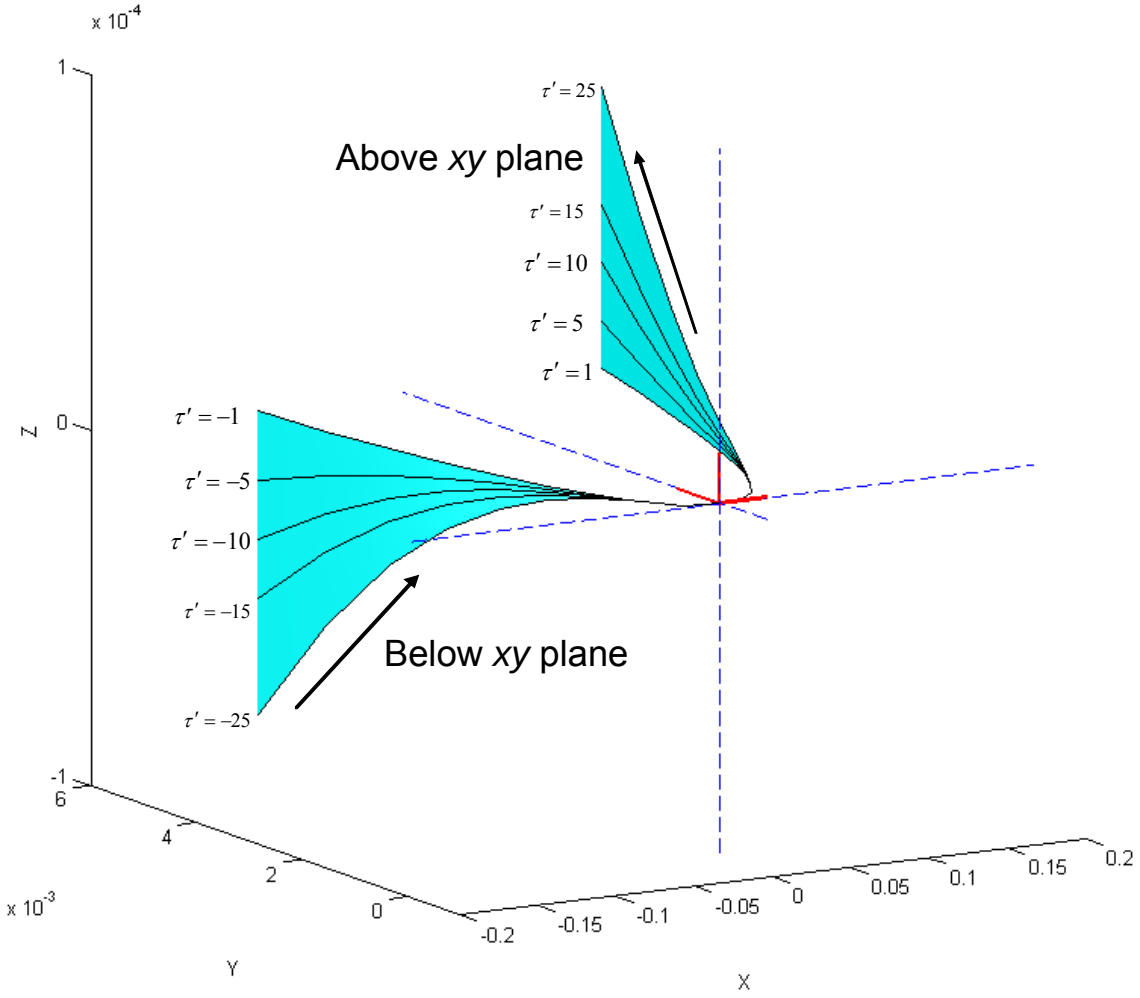


Figure 4.24. Saddle Point 2

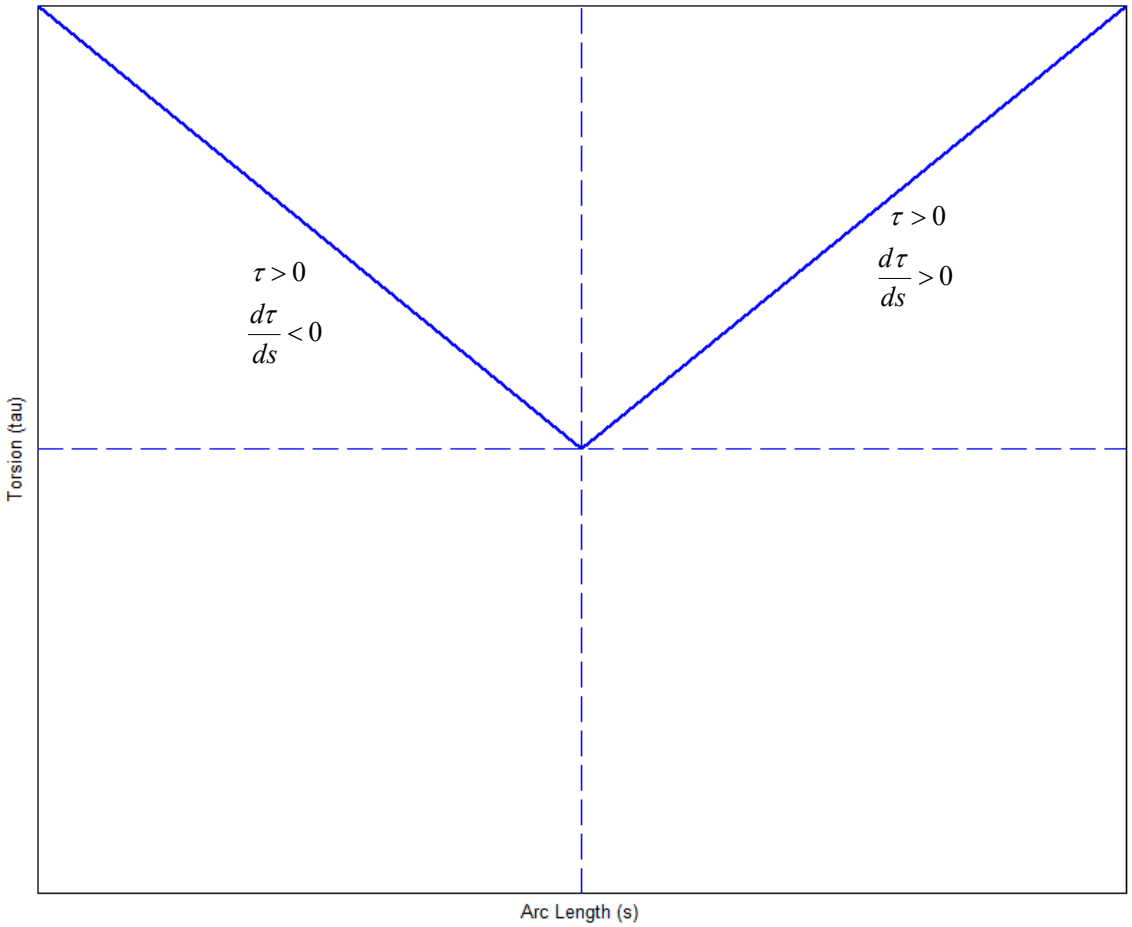


Figure 4.25. Saddle Point 2 Torsion Profile

4.4.4. Spatial Inflection Point

A description of planar inflection points was introduced in Section 4.3.5 of this chapter. As stated before, an inflection point in a planar curve occurs when the sign of the curvature changes. However, by definition, curvature is always positive in spatial curves. Thus, an inflection point cannot be defined simply by enforcing a constraint of $\kappa = 0$ at the desired location. Section 4.3.5 also showed that at an inflection point the normal vector flips direction (i.e. inverts). This provides a good geometric description of what occurs at an inflection point. Because a curve tends to bend in the direction of the normal vector, an inflection point represents a point where the curve begins to bend in a different

direction. For example, by flipping the direction of the normal vector at the local frame, a spatial saddle could be created as in Figure 4.26. Similarly, Figure 4.27 shows another kind of saddle point created using an inflection point. These plots differ from the saddle points shown before in Figure 4.22 and Figure 4.24 in that the approaching branch bends in the opposite direction of the normal vector. Thus, an inflection point can be used as an additional constraint on top of curvature and torsion to control the direction of bending in the osculating plane.

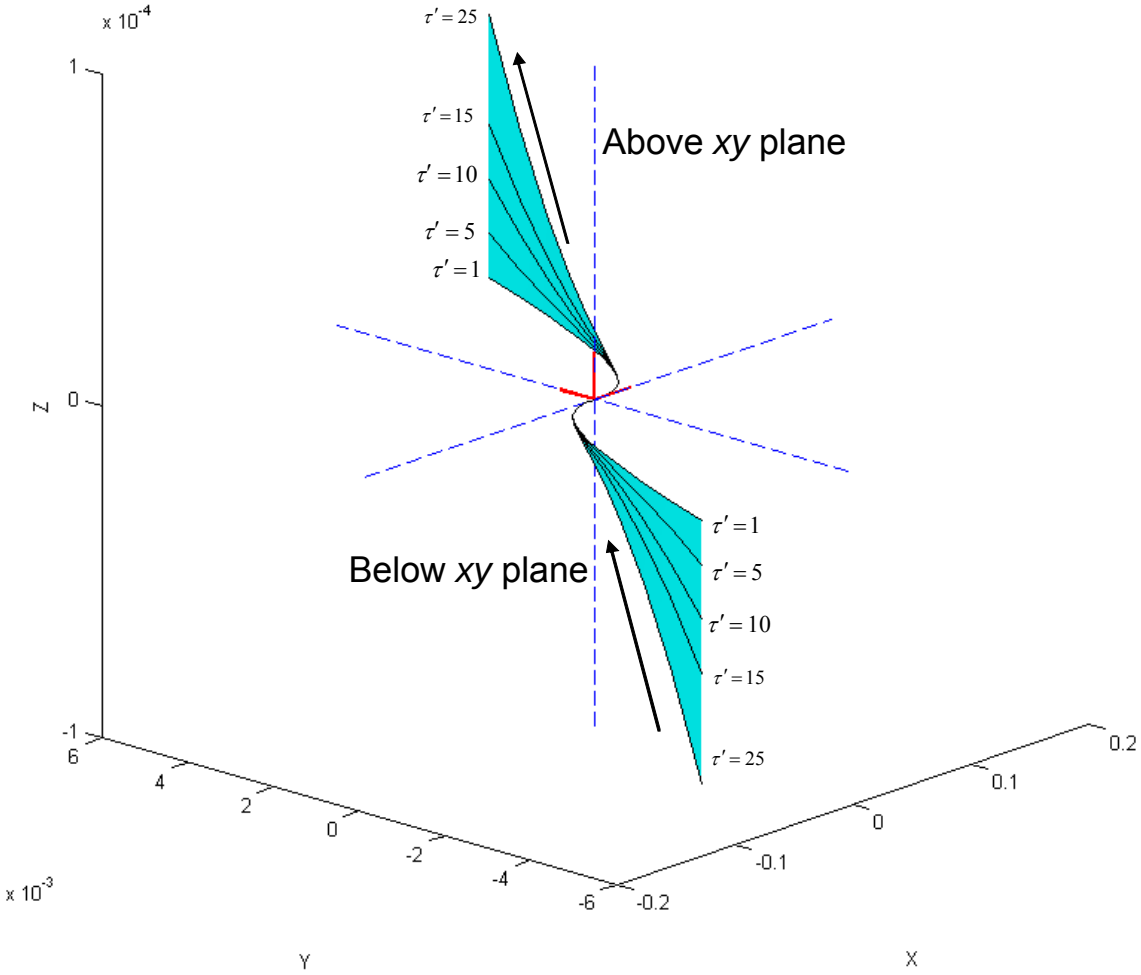


Figure 4.26. Spatial Saddle with an Inflection Point

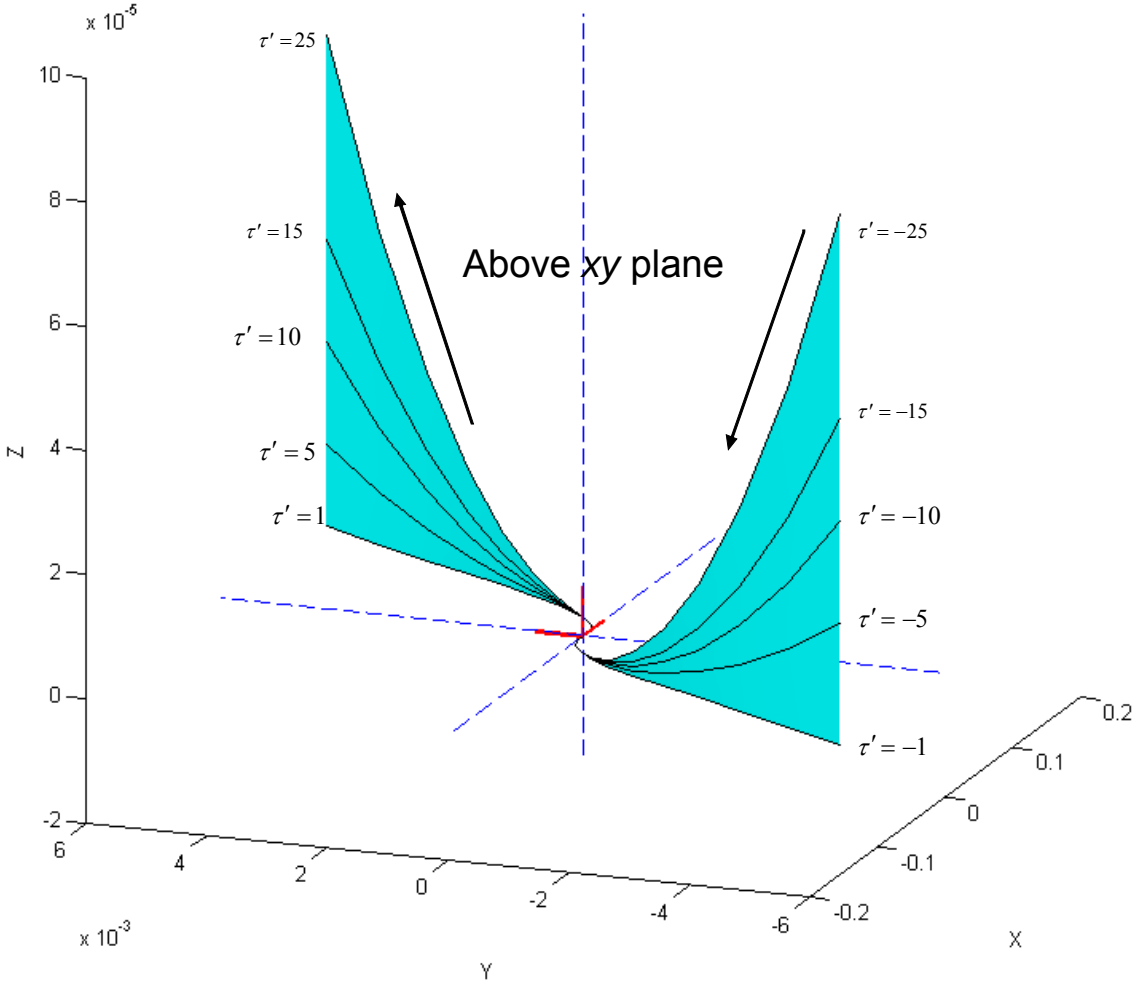


Figure 4.27. Spatial Saddle II with an Inflection Point

4.5. SUMMARY

This chapter presented a study of the relationship between geometric shapes and their higher-order properties. This began with an investigation of simple planar shapes such as circles, parabolas, and ellipses. This analysis began by defining these shapes in their implicit forms and defining parameters to describe families of curves. Then, closed-form solutions for curvature were developed in terms of these parameters. While these solutions were often complex, they will allow a user to define the local shape of these geometries at any point in terms of curvature.

Then, an investigation of the properties of spatial curves was presented. Due to the complex nature of spatial curves, this analysis looked at the effects of curvature and torsion on spatial curves by actually generating the local geometry of curves in terms of curvature and torsion values. This provides a unique perspective on these higher-order properties. Then, several example spatial geometric shapes were presented and studied with the goal of defining curvature and torsion based constraints.

The next step in this research is to develop a curve generation technique that can utilize these constraints based on curvature and torsion. This will involve two main steps. First, constraints based on curvature and torsion need to be formulated as constraints on parametric descriptions of curves (e.g. $\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^n x}{du^n}, \frac{d^n y}{du^n}, \frac{d^n z}{du^n}$). Once these constraints have been formulated, methods of trajectory blending (such as trapezoidal motion profiles) can be investigated to generate the overall motions.

5. CHAPTER FIVE

Path Generation using Geometric Constraints

In the previous chapter, the intrinsic geometric properties of curvature and torsion were examined. This was done by studying the affect of these properties on the local geometry of a curve to better understand their physical meanings. This chapter will build on this understanding by showing how to convert these geometric constraints into parametric constraints ($\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^n x}{du^n}, \frac{d^n y}{du^n}, \frac{d^n z}{du^n}$) that can be used to define trajectories. Once these constraints have been formulated, the individual trajectories for $\left[x(u), y(u), z(u)\right]$ can be developed.

This process will first begin by defining coordinate frames in space (a position and orientation). This approach makes sense for robotic motion planning for several reasons. First, simulation environments and CAD models will come with attached frames at key points of interest (potential interaction points). An example of a surgical work cell with defined coordinate frames is shown in Figure 5.1. The frames in this work cell are sometimes attached to specific objects (e.g. approach and grab points for tools or surgical trays) and sometimes used for defined way points for global motions. Second, a coordinate frame provides a natural extension to rotational motion planning. After a frame has been defined, the geometric constraints based on trajectory curvature and torsion can be defined at each point. Then, these geometric constraints can be converted into parametric constraints. These parametric constraints can then be blended together using a variety of techniques. The following section will provide a brief description of how these parametric constraints could be blended to form spatial curves. Then, the following sections will describe the mathematical formulations of these constraints.

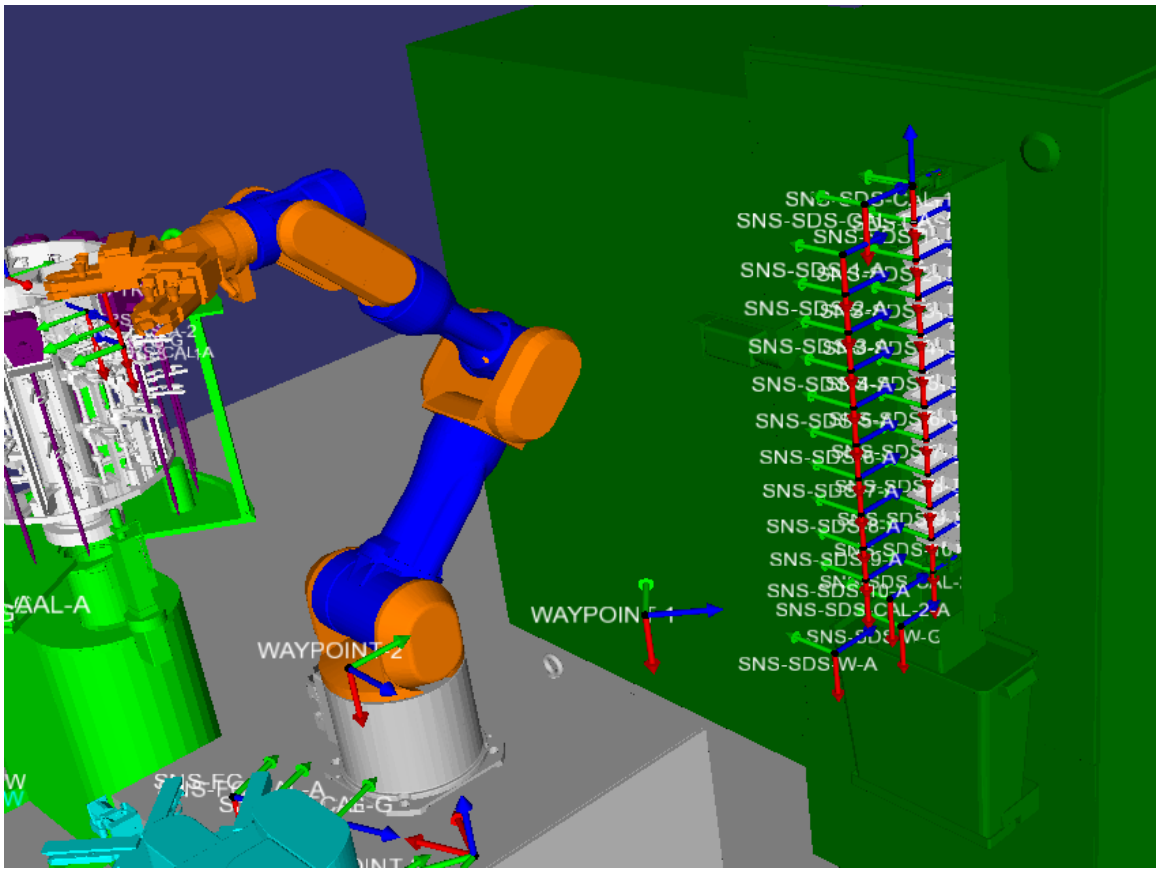


Figure 5.1. Robotic Workcell with Defined Frames of Interest

5.1. INTRODUCTION

As stated before, the main goal of this chapter is to convert geometric constraints into parametric constraints. However, it is useful to first introduce how these parametric constraints can be used as the following sections contain several examples. Suppose the parametric constraints for the *x* coordinate shown in Table 5.1 were provided. There are many different methods to interpolate between these higher-order constraints. For example, Figure 5.2 show these constraints being met using a 3rd order trapezoidal profile. This method basically starts by defining a trapezoid in some higher-derivative and then integrating up to meet the various constraints. A more detailed description of

this formulation as well as comparisons between various blending techniques can be found in [38][50].

x_1	$\frac{dx_1}{du}$	$\frac{d^2x_1}{du^2}$	$\frac{d^3x_1}{du^3}$	x_2	$\frac{dx_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^3x_2}{du^3}$
-1.0	-6.0	-30.0	0.0	2.0	3.0	-20.0	0.0

Table 5.1. Sample Parametric Constraints

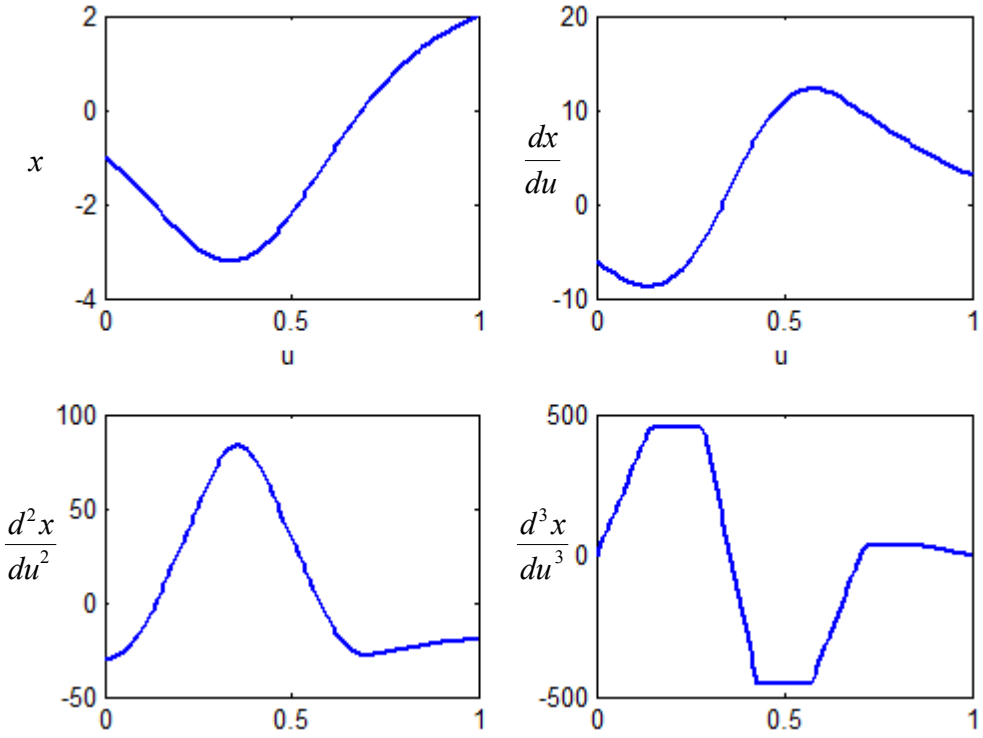


Figure 5.2. Trapezoidal Specification

Thus, all three coordinates (x, y, z) can be planned individually and then combined for the overall path trajectory (Equation 5.1). It should be noted that in this research the main focus is to define/meet these higher-order constraints at the points of interest (i.e. the local geometry of the curves) and little control or optimization over the overall path is provided. However, this motion planning is an important topic for future work. Appendix B includes an initial exploration of this topic.

$$\mathbf{p}(u) = \begin{bmatrix} x(u) & y(u) & z(u) \end{bmatrix} \qquad \textbf{5.1}$$

5.2. FIRST-ORDER PROPERTIES

As mentioned before, this method begins by defining a point (x, y, z) and a frame ($\hat{\mathbf{T}}, \hat{\mathbf{N}}, \hat{\mathbf{B}}$). To define a first-order parametric constraint, the equation of the unit tangent vector (Equation 5.2) can be used.

$$\hat{\mathbf{T}}(\mathrm{u}) = \frac{\frac{d\mathbf{p}}{du}}{\left\| \frac{d\mathbf{p}}{du} \right\|} \qquad \textbf{5.2}$$

The first-order constraint ($\frac{d\mathbf{p}}{du}$) can be easily solved for as shown in Equation 5.3. This equation shows that the geometric constraint can be met for any magnitude of the vector $\frac{d\mathbf{p}}{du}$ as long as it is in the same direction as the unit tangent. This allows for a degree of freedom in defining the first-order parametric constraints.

$$\frac{d\mathbf{p}}{du} = \left\| \frac{d\mathbf{p}}{du} \right\| \hat{\mathbf{T}}(\mathrm{u}) \qquad \textbf{5.3}$$

For example, Table 5.2 shows the various constraints for several different magnitudes of $\frac{d\mathbf{p}}{du}$ [6]. A plot of the resulting curves is shown in Figure 5.3. In this figure, the curves with a higher magnitude of $\frac{d\mathbf{p}}{du}$ tend to have a bias towards the tangent vector on that end.

$\frac{d\mathbf{p}}{du}$	x_1	y_1	$\frac{dx_1}{du}$	$\frac{dy_1}{du}$	x_2	y_2	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$
$\hat{\mathbf{T}}_2$	0.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0
$2\hat{\mathbf{T}}_2$	0.0	0.0	1.0	0.0	1.0	1.0	0.0	2.0
$5\hat{\mathbf{T}}_2$	0.0	0.0	1.0	0.0	1.0	1.0	0.0	5.0

Table 5.2. First-Order Constraints Example

[6] Planar curves are used here for better visualization, but the same concept extends to the spatial domain as well.

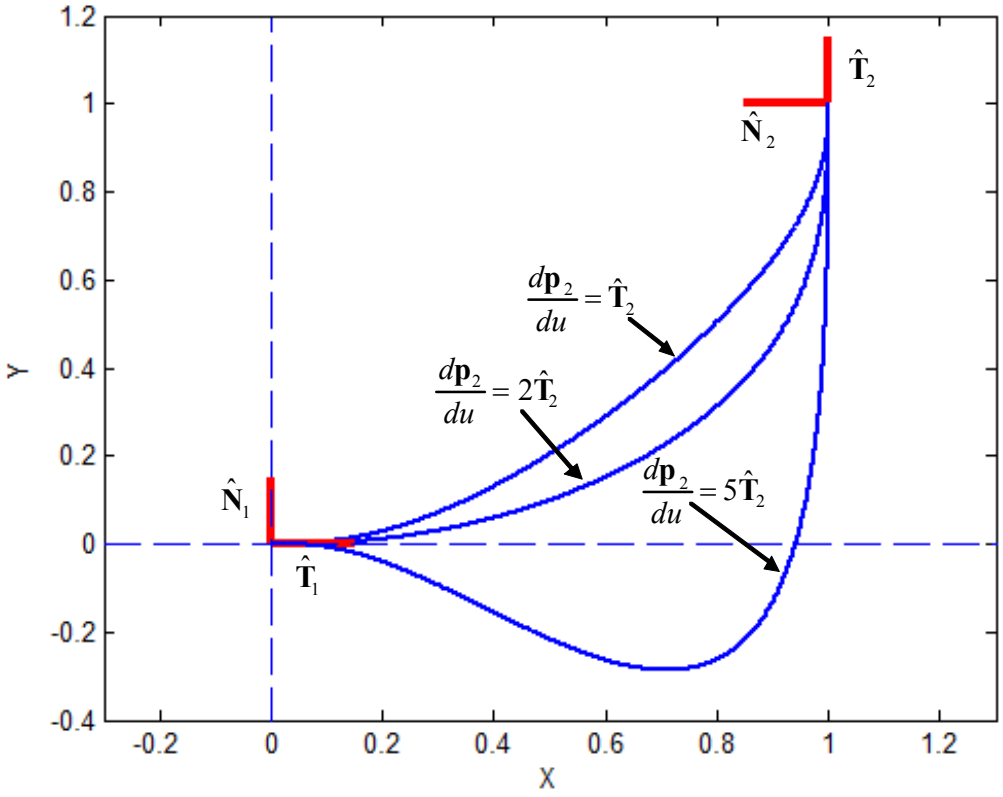


Figure 5.3. First-Order Constraints Example

Table 5.3 shows another set of example constraints, and Figure 5.4 shows the resulting curves. This plot shows similar behavior to the previous example. For the remainder of this chapter, this tangent scale is assumed to be 1.0 for simplicity, and thus $\frac{d\mathbf{p}}{du}$ will always be a unit vector. However, it will be shown later how this parameter can provide an important degree of freedom to the curve design.

$\frac{d\mathbf{p}}{du}$	x_1	y_1	$\frac{dx_1}{du}$	$\frac{dy_1}{du}$	x_2	y_2	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$
$\hat{\mathbf{T}}_2$	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0
$2\hat{\mathbf{T}}_2$	0.0	0.0	1.0	0.0	1.0	1.0	2.0	0.0
$5\hat{\mathbf{T}}_2$	0.0	0.0	1.0	0.0	1.0	1.0	5.0	0.0

Table 5.3. First-Order Constraints Example II

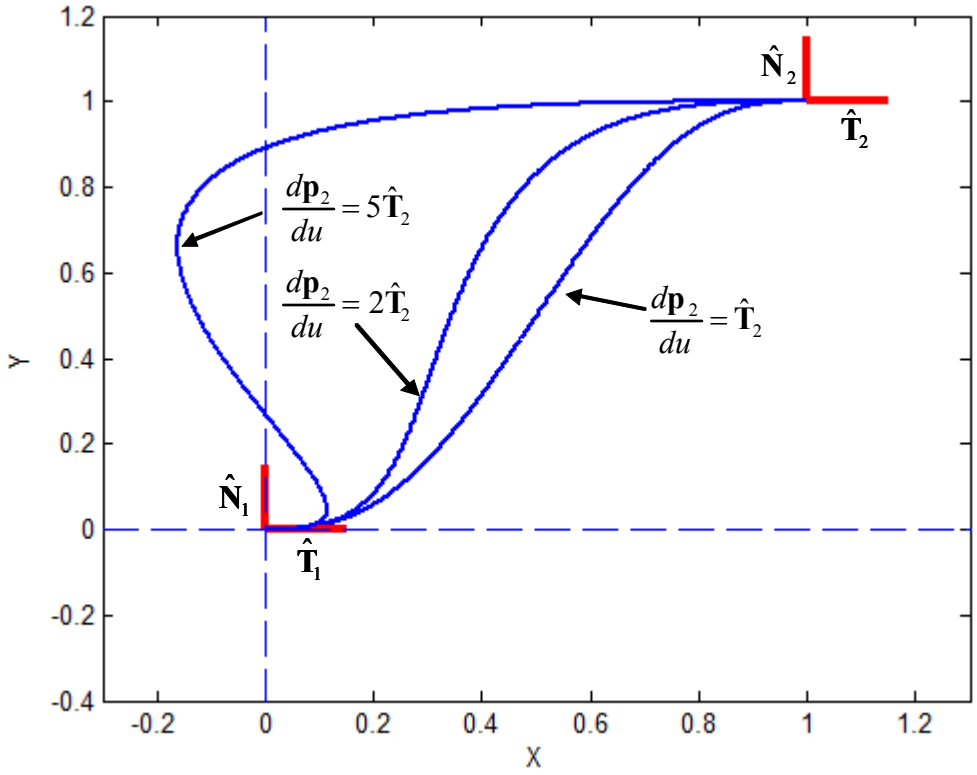


Figure 5.4. First-Order Constraints Example II

5.3. SECOND-ORDER PROPERTIES

Next, the second-order constraints ($\frac{d^2\mathbf{p}}{du^2}$) can be formulated based on the second-order geometric constraints: curvature (Equation 5.4) and the unit normal vector (Equation 5.5).

$$\kappa(u) = \frac{\left\| \frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2} \right\|}{\left\| \frac{d\mathbf{p}}{du} \right\|^3} \qquad \textbf{5.4}$$

$$\hat{\mathbf{N}}(u) = \frac{\frac{d\hat{\mathbf{T}}}{du}}{\left\| \frac{d\hat{\mathbf{T}}}{du} \right\|} \qquad \textbf{5.5}$$

Now, to better define the unit normal vector, the derivative $\frac{d\hat{\mathbf{T}}}{du}$ can be calculated analytically. Using the chain rule, the first step of this derivation is shown in Equation 5.6.

$$\frac{d\hat{\mathbf{T}}}{du} = \frac{d}{du}\left(\frac{\frac{d\mathbf{p}}{du}}{\left\|\frac{d\mathbf{p}}{du}\right\|}\right) = \frac{\frac{d^2\mathbf{p}}{du^2}}{\left\|\frac{d\mathbf{p}}{du}\right\|} + \left[\frac{d}{du}\left(\frac{1}{\left\|\frac{d\mathbf{p}}{du}\right\|}\right)\right]\frac{d\mathbf{p}}{du} \qquad \textbf{5.6}$$

Now, the derivative in the second term can be expanded using the quotient rule as in Equation 5.7. Finally, the derivative of the magnitude $\left\|\frac{\mathbf{d}p}{du}\right\|$ can be evaluated as in Equation 5.8.

$$\frac{d}{du}\left(\frac{1}{\left\|\frac{d\mathbf{p}}{du}\right\|}\right) = \frac{d}{du}\left\|\frac{d\mathbf{p}}{du}\right\|^{-1} = -\frac{\frac{d}{du}\left\|\frac{\mathbf{d}p}{du}\right\|}{\left\|\frac{\mathbf{d}p}{du}\right\|^2} \qquad \textbf{5.7}$$

$$\frac{d}{du}\left\|\frac{d\mathbf{p}}{du}\right\| = \frac{d}{du}\left[\left(\frac{d\mathbf{p}}{du}\cdot\frac{d\mathbf{p}}{du}\right)^{\frac{1}{2}}\right] = \frac{1}{2}\left(\frac{d\mathbf{p}}{du}\cdot\frac{d\mathbf{p}}{du}\right)^{-\frac{1}{2}}\left[2\frac{d\mathbf{p}}{du}\cdot\frac{d^2\mathbf{p}}{du^2}\right] = \frac{\frac{d\mathbf{p}}{du}\cdot\frac{d^2\mathbf{p}}{du^2}}{\left\|\frac{d\mathbf{p}}{du}\right\|} \qquad \textbf{5.8}$$

Plugging all of these results back into the original equation, the equation for $\frac{d\hat{\mathbf{T}}}{du}$ can be formulated as in Equation 5.9.

$$\frac{d\hat{\mathbf{T}}}{du} = \frac{\left\|\frac{d\mathbf{p}}{du}\right\|^2\frac{d^2\mathbf{p}}{du^2} - \left(\frac{d^2\mathbf{p}}{du^2}\cdot\frac{d\mathbf{p}}{du}\right)\frac{d\mathbf{p}}{du}}{\left\|\frac{d\mathbf{p}}{du}\right\|^3} \qquad \textbf{5.9}$$

Now, this equation can be related back to curvature using the relationship (from the Serret-Frenet formulas) $\frac{d\hat{\mathbf{T}}}{du} = \left\|\frac{d\mathbf{p}}{du}\right\|\kappa\hat{\mathbf{N}}$. To further simplify this relationship, $\frac{d\mathbf{p}}{du}$ and $\frac{d^2\mathbf{p}}{du^2}$ can be assumed to be perpendicular, so that the right-hand side of Equation 5.9

becomes equal to zero. This leads to the expression for the second-order parametric constraints ($\frac{d^2\mathbf{p}}{du^2}$) in terms of curvature and unit normal vector shown in Equation 5.10.

$$\frac{d^2\mathbf{p}}{du^2} = \left\|\frac{d\mathbf{p}}{du}\right\|^2 \kappa\hat{\mathbf{N}} \qquad \textbf{5.10}$$

Now, using this relationship, a curve can be defined to pass through a point with a given curvature. For example, consider a path plan that is provided that must pass through three points with a specified curvature at the middle point (x_2, y_2). Table 5.4 shows how the first and second derivatives could be defined to satisfy three different values for curvature. A plot of these three curves is shown in Figure 5.5. As expected, the higher values of curvature generate a sharper bend (locally) around the second point. As before, the overall shape of the curve is not being controlled as this work is focused on defining the local geometry.

	x_2	y_2	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$
$\kappa = 1$	1.0	1.0	0.0	1.0	1.0	0.0
$\kappa = 5$	1.0	1.0	0.0	1.0	5.0	0.0
$\kappa = 15$	1.0	1.0	0.0	1.0	15.0	0.0

Table 5.4. Curvature Constraints Example

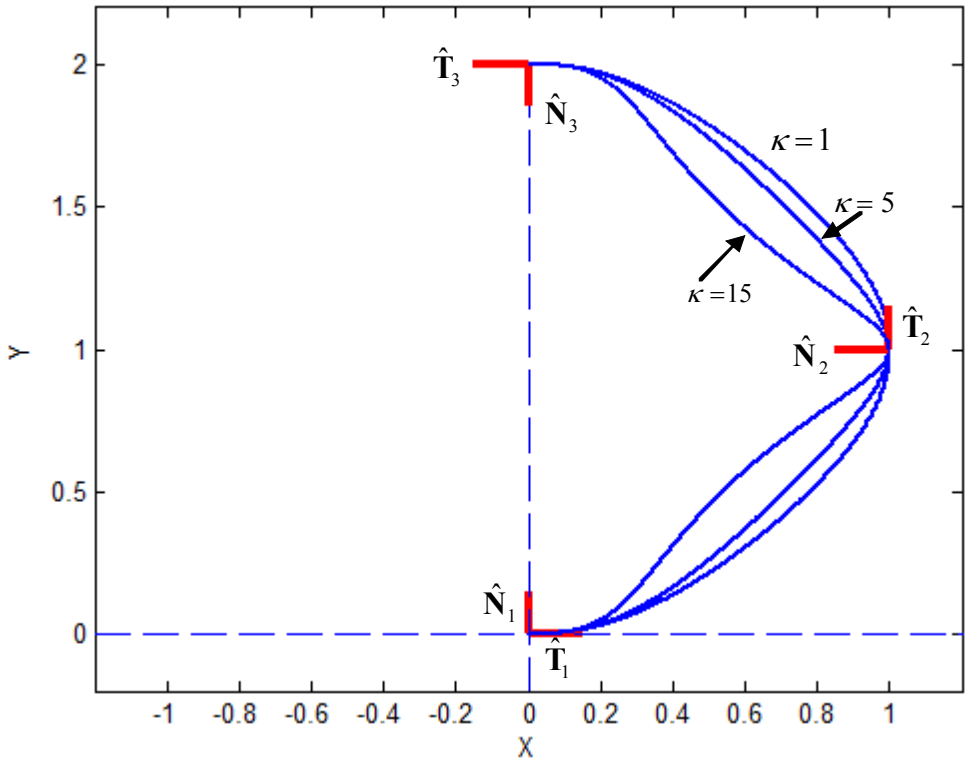


Figure 5.5. Curvature Constraints Example

Another set of example constraints is shown in Table 5.5, and the resulting set of curves is shown in Figure 5.6. Once again, the higher values of curvature lead to sharper bending. Also, this curve illustrates that curvature will always cause a curve to bend around its normal vector.

	x_2	y_2	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$
$\kappa = 1$	1.0	1.0	0.707	0.707	-0.707	0.707
$\kappa = 5$	1.0	1.0	0.707	0.707	-3.5339	3.5339
$\kappa = 15$	1.0	1.0	0.707	0.707	-10.6018	10.6018

Table 5.5. Curvature Constraints Example II

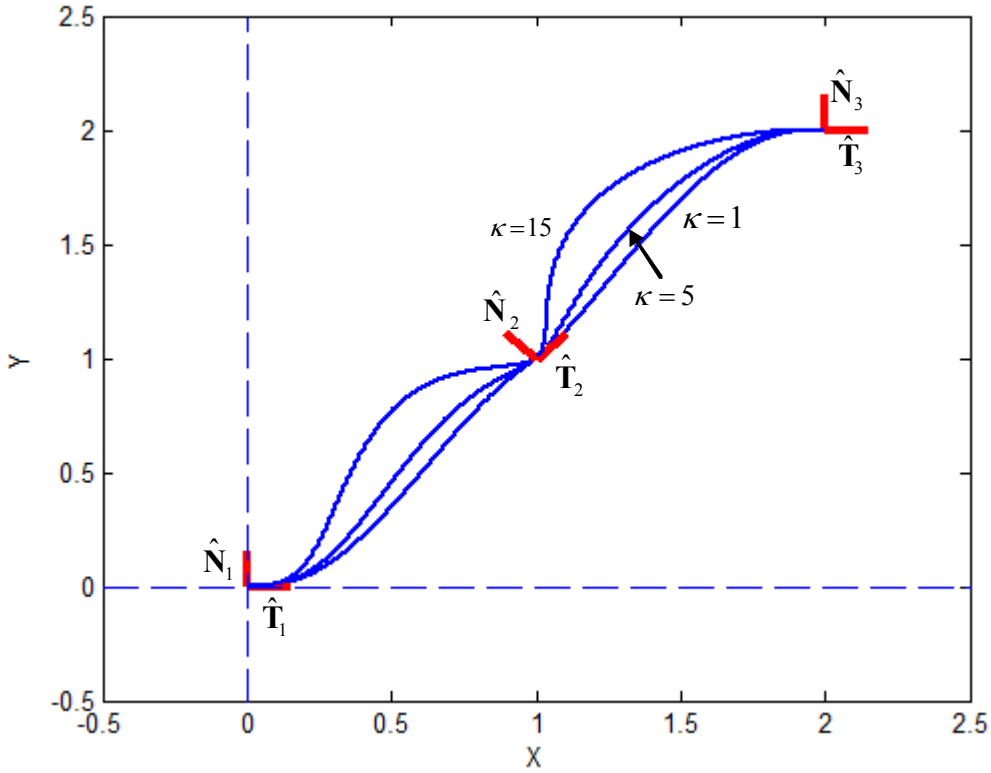


Figure 5.6. Curvature Constraints Example II

5.4. THIRD-ORDER PROPERTIES

With $\frac{d\mathbf{p}}{du}$ and $\frac{d^2\mathbf{p}}{du^2}$ now defined by the unit tangent, unit normal, and curvature, the third-order parametric constraints ($\frac{d^3\mathbf{p}}{du^3}$) can be defined. The first third-order property to be looked at is torsion, shown in vector form in Equation 5.11. This equation shows that torsion involves coupling between the first, second, and third order parametric constraints. However, the first and second order properties are already defined and can be considered known. Thus, this equation need only be solved for $\frac{d^3\mathbf{p}}{du^3}$.

$$\tau(u) = \frac{\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du}\right) \cdot \frac{d^3\mathbf{p}}{du^3}}{\left\| \frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2} \right\|^2} \quad \textbf{5.11}$$

Expanding this equation out in terms of the individual components leads to Equation 5.12. Then, by collecting like terms, the individual third-order properties can be isolated as shown in Equation 5.13.

$$\tau(u) = \frac{\left(y'z''x''' - y''z'x'''\right) + \left(z'x''y''' - x'z''y'''\right) + \left(x'y''z''' - y'x''z'''\right)}{\left(y'z'' - y''z'\right)^2 + \left(z'x'' - x'z''\right)^2 + \left(x'y'' - y'x''\right)^2} \quad \textbf{5.12}$$

$$\tau(u) = \frac{x'''\left(y'z'' - y''z'\right) + y'''\left(x''z' - x'z''\right) + z'''\left(x'y'' - x''y'\right)}{\left(y'z'' - y''z'\right)^2 + \left(x''z' - x'z''\right)^2 + \left(x'y'' - x''y'\right)^2} \quad \textbf{5.13}$$

However, because the first and second-order properties have already been defined, everything except x''', y''', and z''' in this equation can be considered constant. This equation can then be rewritten as shown in 5.14 where a_0, b_0, and c_0 are constants defined by the first and second order parametric constraints. This leads to a simple linear system that can be solved for x''', y''', and z'''. The results in this chapter are obtained by performing a pseudo-inverse to determine the value of torsion.

$$\tau(u) = a_0 x''' + b_0 y''' + c_0 z''' \quad \textbf{5.14}$$

Now, parametric constraints can be developed up to the third-order using geometric constraints on both curvature and torsion. Consider the example constraints shown in Table 5.6. These constraints involve passing through three frames with a defined curvature and torsion at each. Using the process described above, the parametric constraints can be calculated as shown in Table 5.7.

	x	y	z	$\hat{\mathbf{T}}$	$\hat{\mathbf{N}}$	$\hat{\mathbf{B}}$	κ	τ
$\mathbf{p}_1$	0.0	0.0	0.0	[1,0,0]	[0,1,0]	[0,0,1]	0.0	0.0
$\mathbf{p}_2$	2.0	1.0	1.0	[0,0.707,-0.707]	[-1,0,0]	[0,0.707,0.707]	1.0	10.0
$\mathbf{p}_3$	1.0	3.0	-1.0	[0,1,0]	[-1,0,0]	[0,0,1]	0.0	0.0

Table 5.6. Example Geometric Constraints with Torsion

i	x_i	y_i	z_i	$\frac{dx_i}{du}$	$\frac{dy_i}{du}$	$\frac{dz_i}{du}$	$\frac{d^2x_i}{du^2}$	$\frac{d^2y_i}{du^2}$	$\frac{d^2z_i}{du^2}$	$\frac{d^3x_i}{du^3}$	$\frac{d^3y_i}{du^3}$	$\frac{d^3z_i}{du^3}$
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0	1.0	1.0	0.0	0.707	-.707	-1.0	0.0	0.0	0.0	7.0679	7.0679
3	1.0	3.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5.7. Example Calculated Parametric Constraints

Once these constraints are defined, the individual trajectories for $x(u)$, $y(u)$, and $z(u)$ can be calculated. Figure 5.7 shows the overall spatial trajectory using the calculated values. Figure 5.8 and Figure 5.9 show the curvature and torsion profiles for this trajectory. It can be seen that the curvature and torsion have the correct values at the defined point; however, there is little control over these parameters in between the defined points.

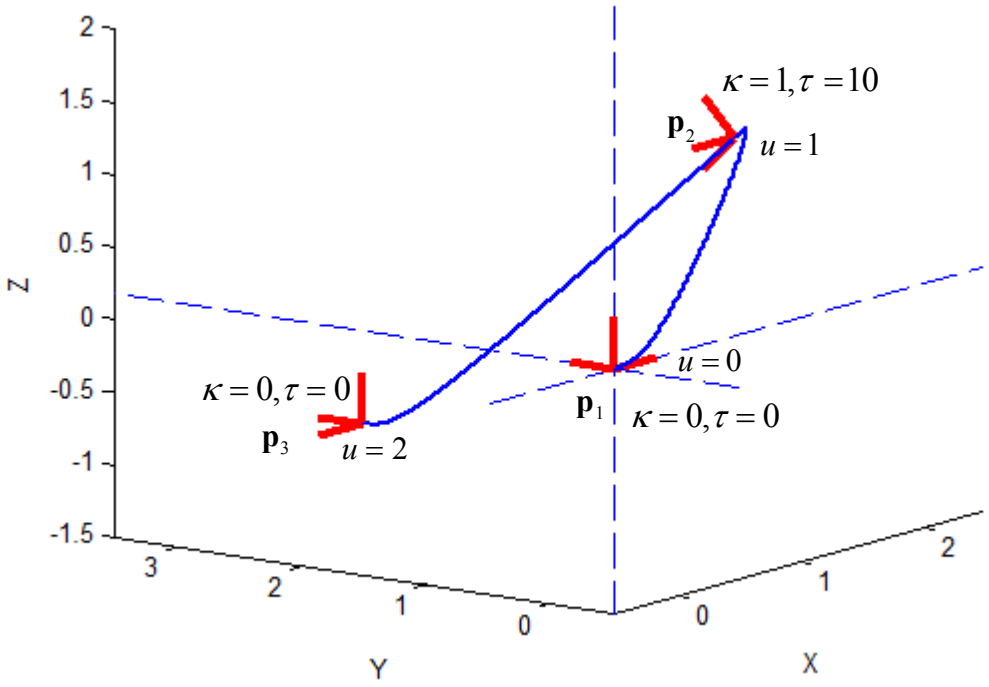


Figure 5.7. Spatial Trajectory with Torsion Constraints

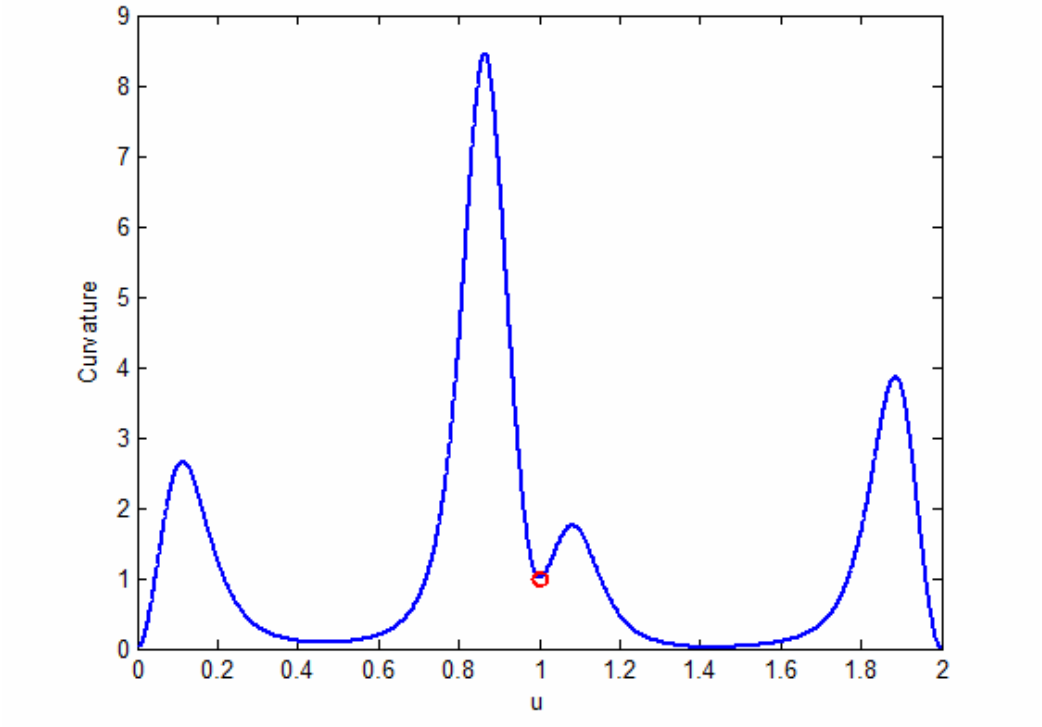


Figure 5.8. Curvature Profiles

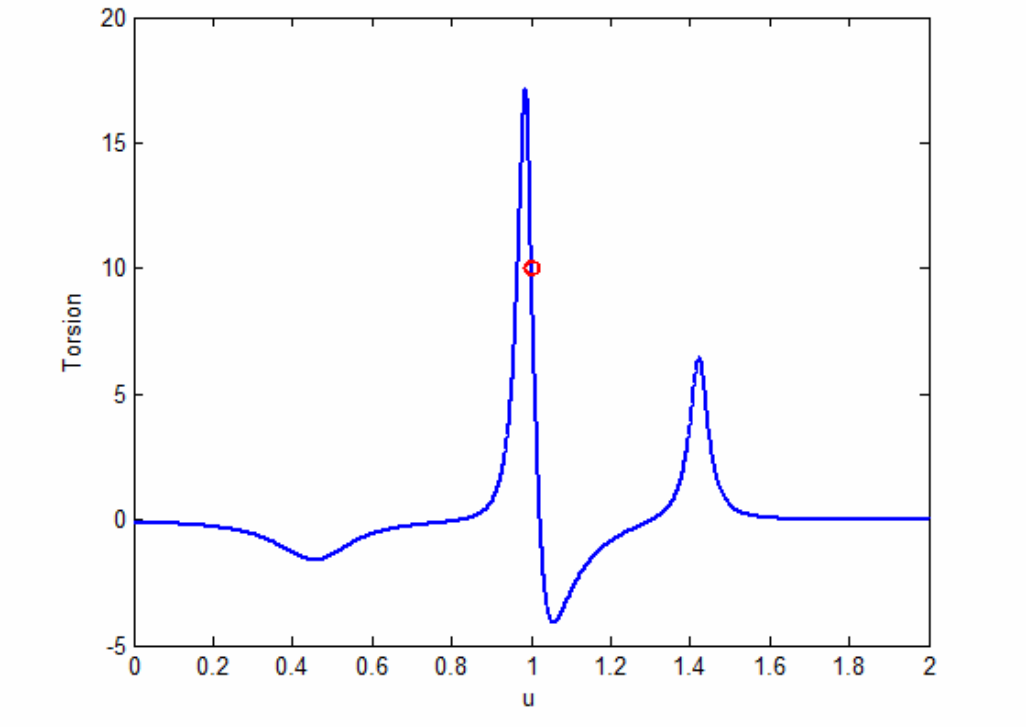


Figure 5.9. Torsion Profile

Another property that depends on the third-order parametric constraints is the derivative of curvature as shown in Equation 5.15 [58]. Because the first and second order properties have been defined to be perpendicular, the second term in the numerator will disappear and this equation will reduce to Equation 5.16.

$$\frac{d\kappa}{du} = \frac{\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right) \cdot \left(\frac{d\mathbf{p}}{du} \times \frac{d^3\mathbf{p}}{du^3}\right) - 3\kappa^2 \left(\frac{d\mathbf{p}}{du} \cdot \frac{d^2\mathbf{p}}{du^2}\right) \left\|\frac{d\mathbf{p}}{du}\right\|^4}{\kappa \left\|\frac{d\mathbf{p}}{du}\right\|^6} \qquad \textbf{5.15}$$

$$\frac{d\kappa}{du} = \frac{\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right) \cdot \left(\frac{d\mathbf{p}}{du} \times \frac{d^3\mathbf{p}}{du^3}\right)}{\kappa \left\|\frac{d\mathbf{p}}{du}\right\|^6} \qquad \textbf{5.16}$$

Now, the values of x''', y''', and z''' need to be isolated, so that their values can be calculated. First, the term $\left(\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right)$ can be replaced with constant values, because these terms have already been defined as shown in 5.17.

$$\left(\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right)=\begin{bmatrix}\left(y'z''-z'y''\right)\\ \left(x''z'-x'z''\right)\\ \left(x'y''-y'x''\right)\end{bmatrix}=\begin{bmatrix}C_1\\ C_2\\ C_3\end{bmatrix} \qquad \textbf{5.17}$$

Then, the entire expression can be expanded and Equation 5.18 can be reached by collecting the third-order terms.

$$\left(\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right)\cdot\left(\frac{d\mathbf{p}}{du}\times\frac{d^3\mathbf{p}}{du^3}\right)=x'''\left(C_2z'-C_3y'\right)+y'''\left(C_3x'-C_1z'\right)+z'''\left(C_1y'-C_2x'\right) \qquad \textbf{5.18}$$

Finally, the above equation can be combined with Equation 5.16 to come up with the final result shown in Equation 5.19 where a_1, b_1, and c_1 can be considered constants.

$$\frac{d\kappa}{du}\kappa\left\|\frac{d\mathbf{p}}{du}\right\|^6=\begin{bmatrix}a_1 & b_1 & c_1\end{bmatrix}\begin{bmatrix}x'''\\ y'''\\ z'''\end{bmatrix} \qquad \textbf{5.19}$$

Now, this constraint for $\frac{d\kappa}{du}$ can be combined with the torsion constraint defined before in Equation 5.14 to create the system of linear equations shown in Equation 5.20. With two constraints and three unknowns, this system can be solved to yield solutions[7] for $\frac{d^3\mathbf{p}}{du^3}$ that can satisfy constraints on both torsion and the derivative of curvature.

$$\begin{bmatrix}\tau\\ \frac{d\kappa}{du}\kappa\left\|\frac{d\mathbf{p}}{du}\right\|^6\end{bmatrix}=\begin{bmatrix}a_0 & b_0 & c_0\\ a_1 & b_1 & c_1\end{bmatrix}\begin{bmatrix}x'''\\ y'''\\ z'''\end{bmatrix} \qquad \textbf{5.20}$$

An example path description is shown in Table 5.8. This path plan defines a trajectory through three points with constraints defined on the curvature, derivative of curvature, and torsion at the second point. Table 5.9 shows the parametric constraints

[7] As before, a pseudo-inverse is used to evaluate this equation.

calculated from these values. The resulting path is shown in Figure 5.10. Plots of the higher-order properties can be seen in Figure 5.11, Figure 5.12, and Figure 5.13.

	x	y	z	$\hat{\mathbf{T}}$	$\hat{\mathbf{N}}$	$\hat{\mathbf{B}}$	κ	$\frac{d\kappa}{du}$	τ
$\mathbf{p}_1$	0.0	0.0	0.0	[1,0,0]	[0,1,0]	[0,0,1]	0.0	0.0	0.0
$\mathbf{p}_2$	1.0	1.0	1.0	[0,0.707,0.707]	[-1,0,0]	[0,-0.707,0.707]	3.0	-20.0	-10.0
$\mathbf{p}_3$	1.5	2.0	0.0	[0,1,0]	[-1,0,0]	[0,0,1]	0.0	0.0	0.0

Table 5.8. Geometric Constraints with Torsion and Derivative of Curvature

i	x_i	y_i	z_i	$\frac{dx_i}{du}$	$\frac{dy_i}{du}$	$\frac{dz_i}{du}$	$\frac{d^2x_i}{du^2}$	$\frac{d^2y_i}{du^2}$	$\frac{d^2z_i}{du^2}$	$\frac{d^3x_i}{du^3}$	$\frac{d^3y_i}{du^3}$	$\frac{d^3z_i}{du^3}$
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	1.0	1.0	0.0	0.707	0.707	-3.0	0.0	0.0	19.994	21.204	-21.204
3	1.5	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5.9. Example Calculated Parametric Constraints

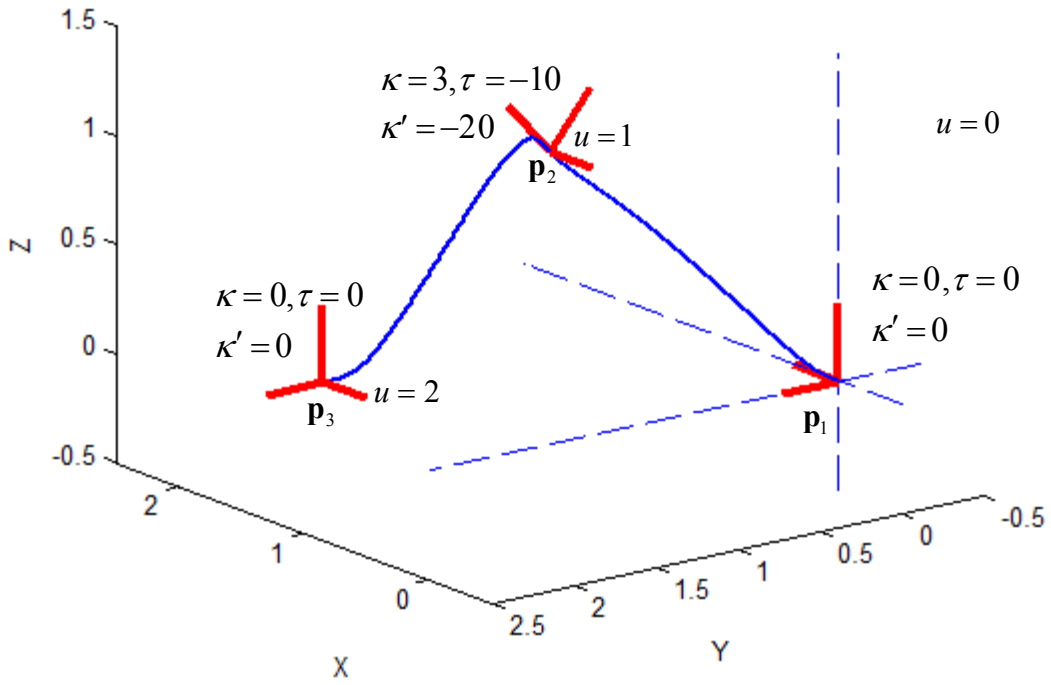


Figure 5.10. Spatial Trajectory with τ and κ' Constraints

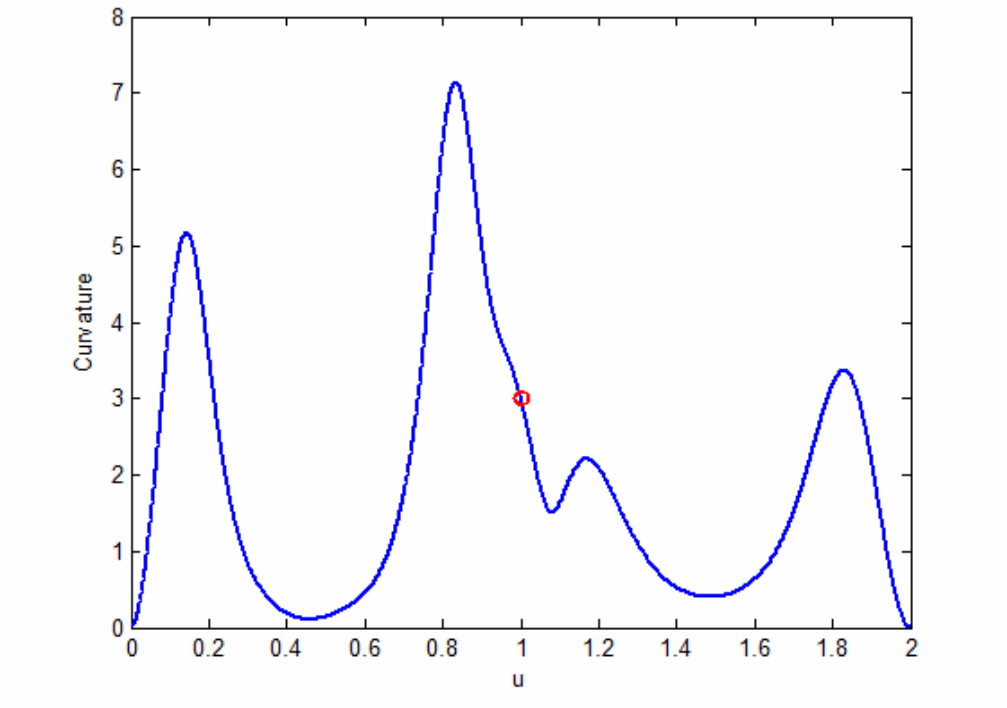


Figure 5.11. Curvature Profile

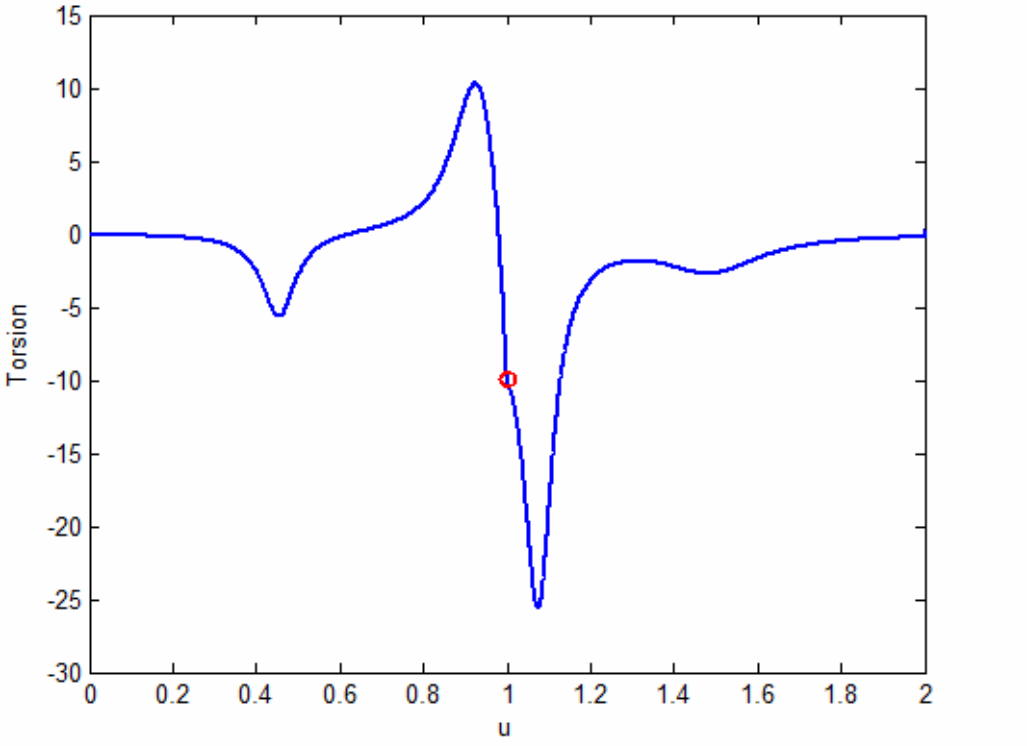


Figure 5.12. Torsion Profile

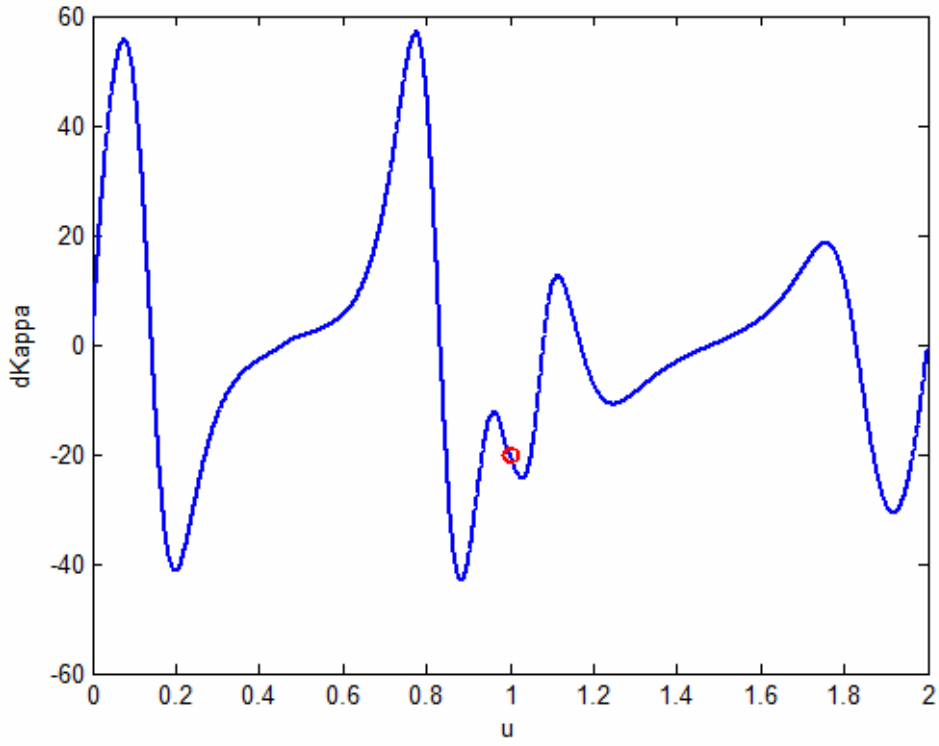


Figure 5.13. Derivative of Curvature Profile

5.5. FOURTH-ORDER PROPERTIES

Finally, the fourth order property of τ' can be examined. The parametric equation for τ' [58] is shown in Equation 5.21. This equation shows that τ' is a function of the first, second, third, and fourth order parametric derivatives. However, the first through third order constraints have already been defined. Thus, the fourth-order derivative must be isolated and solved. Equation 5.22 shows the "constant" values moved to the left-hand side of the equation.

$$\frac{d\tau}{du} = \frac{\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right) \cdot \frac{d^4\mathbf{p}}{du^4} - 2\tau\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right) \cdot \left(\frac{d\mathbf{p}}{du} \times \frac{d^3\mathbf{p}}{du^3}\right)}{\left\|\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right\|^2} \tag{5.21}$$

$$\frac{d\tau}{du}\left\|\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right\|^2 + 2\tau\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right) \cdot \left(\frac{d\mathbf{p}}{du} \times \frac{d^3\mathbf{p}}{du^3}\right) = \left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2}\right) \cdot \frac{d^4\mathbf{p}}{du^4} \tag{5.22}$$

Now, the $\left(\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right)$ can once again be considered constant, which leads to another simple linear system as shown in Equation 5.23.

$$\frac{d\tau}{du}\left\|\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right\|^2+2\tau\left(\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right)\cdot\left(\frac{d\mathbf{p}}{du}\times\frac{d^3\mathbf{p}}{du^3}\right)=a_0x^{(4)}+b_0y^{(4)}+c_0z^{(4)} \qquad \textbf{5.23}$$

Now, a path plan that meets constraints κ, κ', τ, and τ' can be defined by calculating parametric constraints up the fourth-order. For example, consider the geometric constraints shown in Table 5.10. Using the techniques developed in this chapter, these constraints can be mapped to the parametric constraints shown in Table 5.11. Then, the *x, y,* and *z* trajectories can be developed independently and combined together to retrieve the desired spatial curve $\left(\mathbf{p}(u)=\begin{bmatrix}x(u) & y(u) & z(u)\end{bmatrix}\right)$.

	x	y	z	$\hat{\mathbf{T}}$	$\hat{\mathbf{N}}$	$\hat{\mathbf{B}}$	κ	$\frac{d\kappa}{du}$	τ	$\frac{d\tau}{du}$
$\mathbf{p}_1$	0.0	0.0	0.0	[1,0,0]	[0,-1,0]	[0,0,-1]	0.0	0.0	0.0	0.0
$\mathbf{p}_2$	1.0	1.0	1.0	$\begin{bmatrix}0.707\\0.3536\\0.6124\end{bmatrix}$	$\begin{bmatrix}0.0\\-0.866\\0.5\end{bmatrix}$	$\begin{bmatrix}0.707\\-0.3536\\-0.6124\end{bmatrix}$	3.0	-10.0	-5.0	25.0
$\mathbf{p}_3$	2.0	-2.0	-0.5	[0,-1,0]	[-1,0,0]	[0,0,-1]	0.0	0.0	0.0	0.0

Table 5.10. Geometric Constraints

i	1	2	3	i	1	2	3	i	1	2	3
x_i	0.0	1.0	2.0	y_i	0.0	1.0	-2.0	z_i	0.0	1.0	0.5
$\frac{dx_i}{du}$	1.0	0.0	0.0	$\frac{dy_i}{du}$	0.0	0.707	-1.0	$\frac{dz_i}{du}$	0.0	-0.707	0.0
$\frac{d^2x_i}{du^2}$	0.0	0.0	0.0	$\frac{d^2y_i}{du^2}$	0.0	-2.598	0.0	$\frac{d^2z_i}{du^2}$	0.0	1.5	0.0
$\frac{d^3x_i}{du^3}$	0.0	-10.606	0.0	$\frac{d^3y_i}{du^3}$	0.0	13.962	0.0	$\frac{d^3z_i}{du^3}$	0.0	4.183	0.0
$\frac{d^4x_i}{du^4}$	0.0	123.745	0.0	$\frac{d^4y_i}{du^4}$	0.0	-61.861	0.0	$\frac{d^4z_i}{du^4}$	0.0	-107.142	0.0

Table 5.11. Parametric Constraints

Figure 5.14 shows the resulting spatial curve. As before, it can be difficult to visualize how this curve is moving spatially in a 2D representation. However, this research is more concerned with meeting the defined geometric constraints (i.e. the local geometry around each frame of interest) than the overall shape of the curve. These local geometries were presented visually in the previous chapter. Then, this chapter has shown how to convert these into mathematic constraints that can be used to generate spatial curves. Figure 5.15 through Figure 5.18 show the various geometric properties along the length of the spatial curve. Once again, these properties are shown to interpolate the correct values at their defined points, but are not controlled or well-behaved along the entire path.

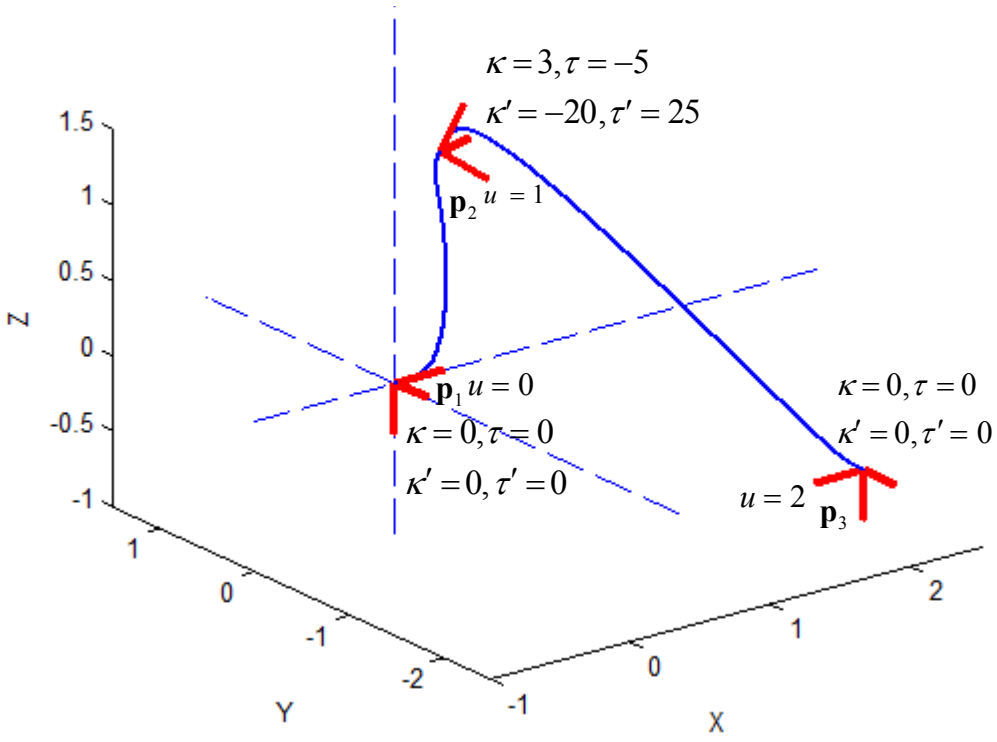


Figure 5.14. Spatial Trajectory

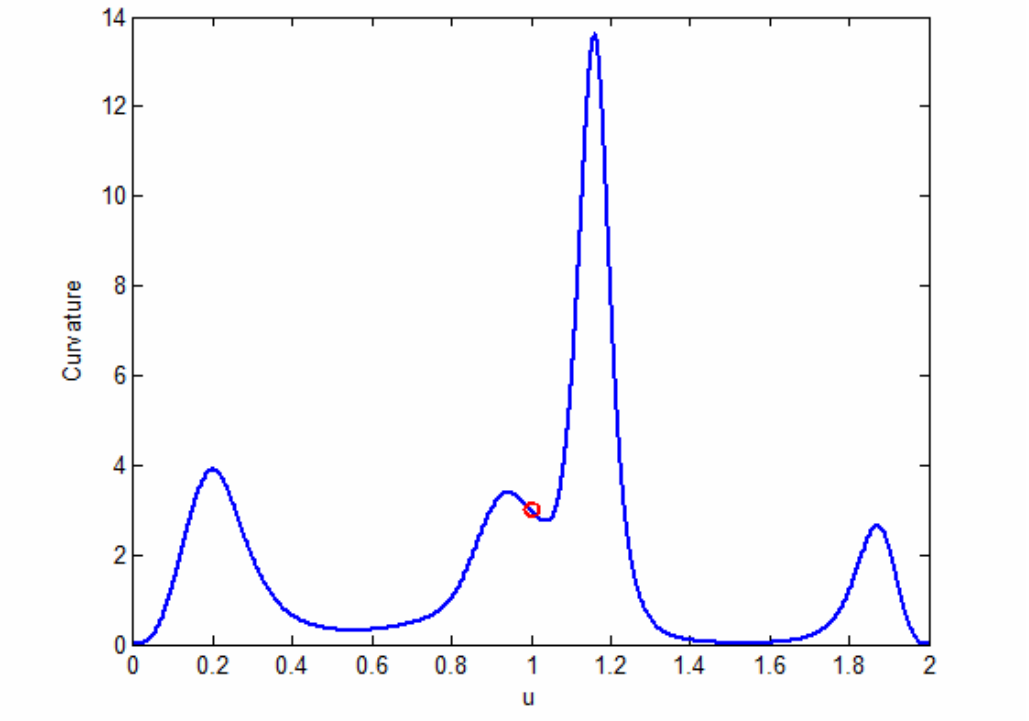


Figure 5.15. Curvature Profile

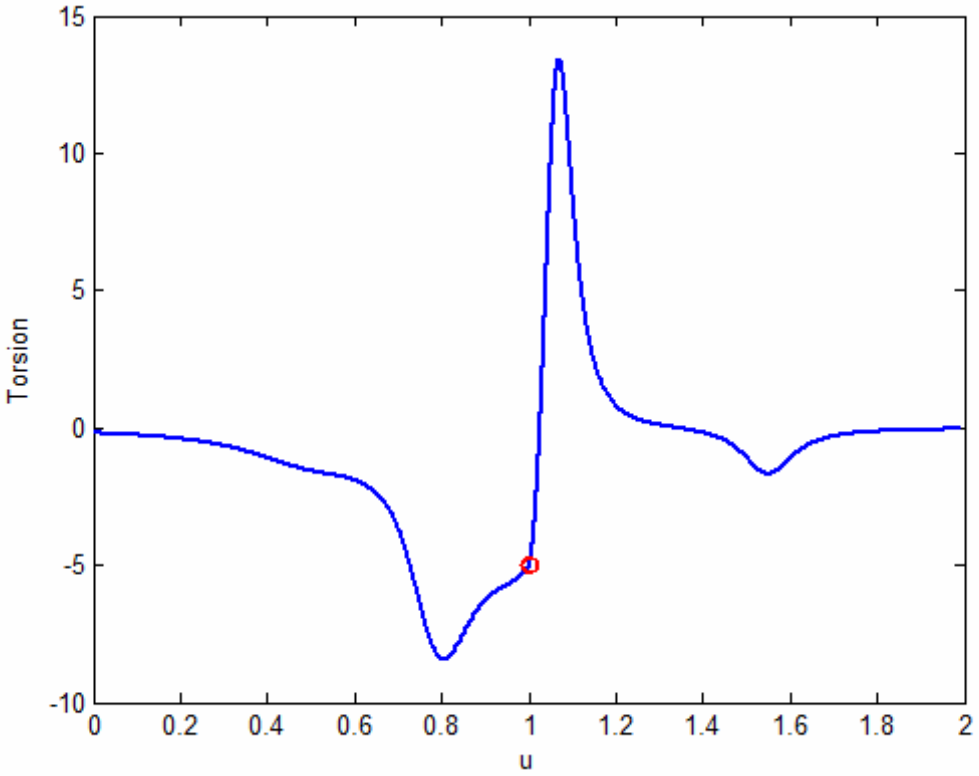


Figure 5.16. Torsion Profile

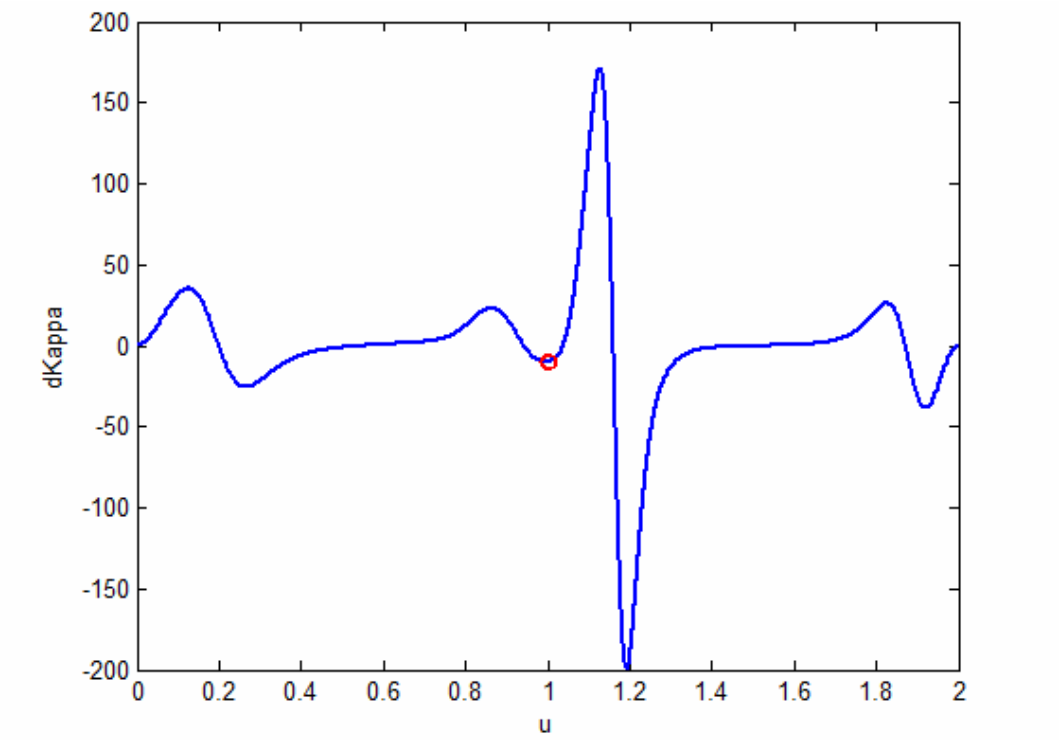


Figure 5.17. Derivative of Curvature Profile

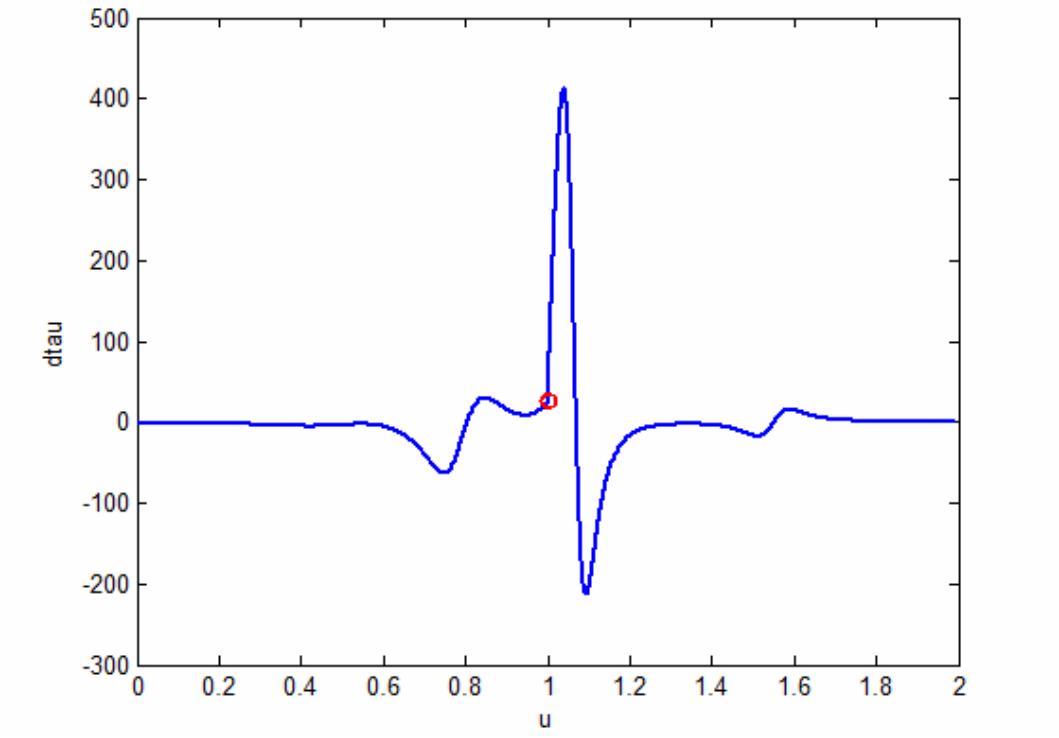


Figure 5.18. Derivative of Torsion Profile

5.6. SPECIAL CASES

In addition to the general shapes that could be generated using curvature and torsion explored in the previous chapter, two special cases were also introduced: cusps and inflection points. In these cases, the geometry of the coordinate frame changes at the point of interest. In the case of an inflection point, the normal vector switches direction. In the case of a cusp, the tangent vector switches directions. This section will describe how to incorporate these special cases into the same procedure presented earlier in this chapter.

5.6.1. Inflection Point

As discussed before in Sections 4.3.5 and 4.4.4, the normal vector inverts at an inflection point. This represents the curve beginning to bend in a different direction. Mathematically, a necessary (but not sufficient) condition for the existence of an inflection point is a curvature of zero. However, when the curvature is zero, the higher-order geometric properties become undefined, and thus the higher-order parametric constraints become indeterminate (i.e. unfixed). One way around this problem is to use the geometric interpretation of an inflection point (the unit normal vector inverting) to induce an inflection point at the desired point. This simply involves using a different normal vector on the approaching and leaving ends of the desired coordinate frame as in Figure 5.19. This curve uses the same exact geometric constraints as the curve in Figure 5.6 except that the middle point is also defined to be an inflection point. This causes the curve to bend in the opposite direction on either side of the frame.

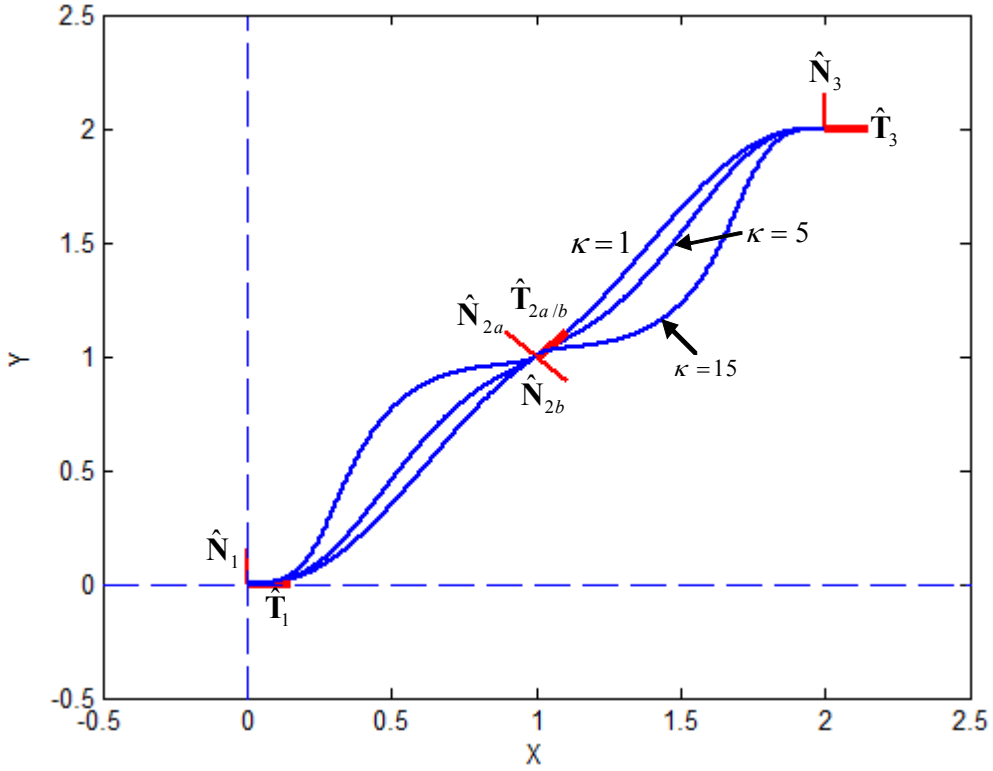


Figure 5.19. Planar Curve with an Inflection Point

To further visualize this, Figure 5.20 shows a zoomed in view of the inflection point. This plot shows the curve begin to bend in the opposite direction at the frame with the curvature dictating how "sharp" the bending is. Thus, the physically meanings of the geometric properties can still be used with this method while still capturing the geometric significance of an inflection point (switching the direction of the curve bending). Table 5.12 and Table 5.13 show the parametric constraints for the approaching and leaving end of the frame, respectively.

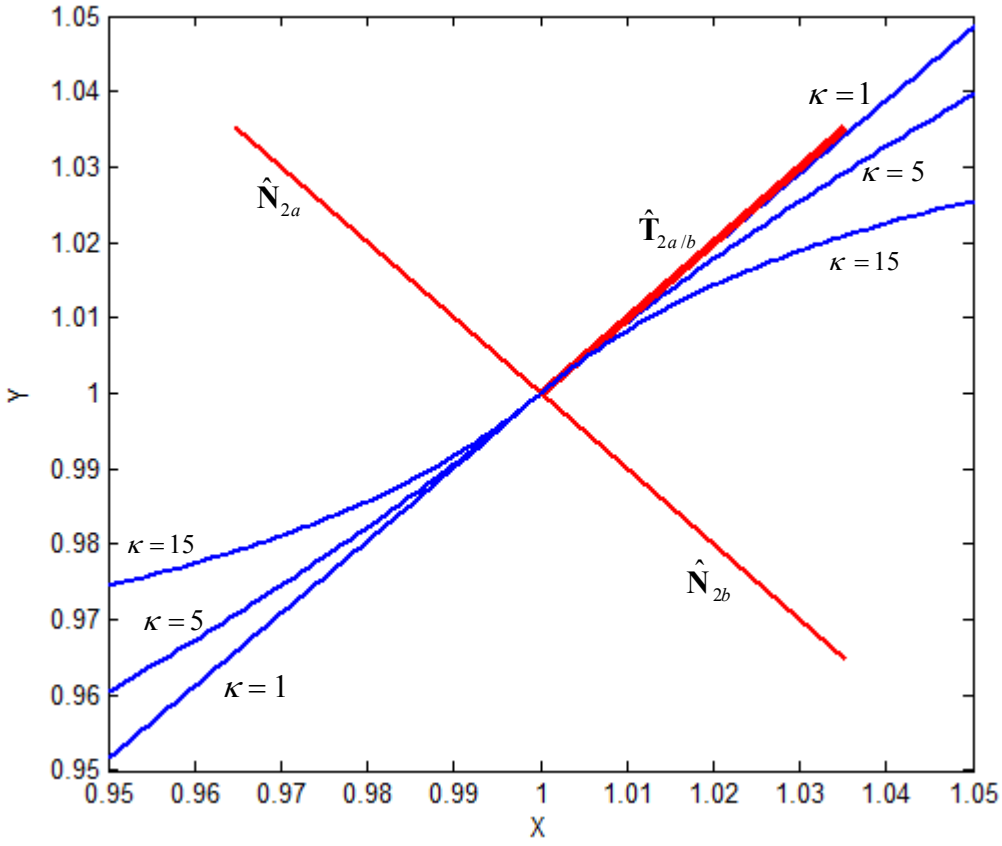


Figure 5.20. Close-up of Inflection Point

	x_{2a}	y_{2a}	$\frac{dx_{2a}}{du}$	$\frac{dy_{2a}}{du}$	$\frac{d^2x_{2a}}{du^2}$	$\frac{d^2y_{2a}}{du^2}$
$\kappa = 1$	1.0	1.0	0.707	0.707	-0.707	0.707
$\kappa = 5$	1.0	1.0	0.707	0.707	-3.536	3.536
$\kappa = 15$	1.0	1.0	0.0	1.0	-10.607	10.607

Table 5.12. Parametric Constraints Approaching Inflection Point

	x_{2b}	y_{2b}	$\frac{dx_{2b}}{du}$	$\frac{dy_{2b}}{du}$	$\frac{d^2x_{2b}}{du^2}$	$\frac{d^2y_{2b}}{du^2}$
$\kappa = 1$	1.0	1.0	0.707	0.707	0.707	-0.707
$\kappa = 5$	1.0	1.0	0.0	-1.0	3.536	-3.536
$\kappa = 15$	1.0	1.0	0.0	-1.0	10.607	-10.607

Table 5.13. Parametric Constraints Leaving Inflection Point

5.6.2. Cusp

As discussed in Section 4.4.2, a necessary condition for a cusp is that the first derivative vector vanishes (i.e. $\frac{d\mathbf{p}}{du} = [0,0,0]$). However, as in the case of an inflection point, this causes the higher-order geometric properties to either go infinite or be undefined. Thus, this mathematical description is not helpful in defining the parametric constraints needed to define the spatial curve. Thus, this method will concentrate on the main geometric significance of a cusp: the inversion of the unit tangent vector. Similar to the method used for an inflection point, this method will simply invert the unit tangent at the desired cusp point[8]. An example of this is shown in Figure 5.21. This figure shows a cusp point with three different curvatures defined around it.

[8] The unit normal vector may also be inverted at this point depending on the desired direction of bending for the curve.

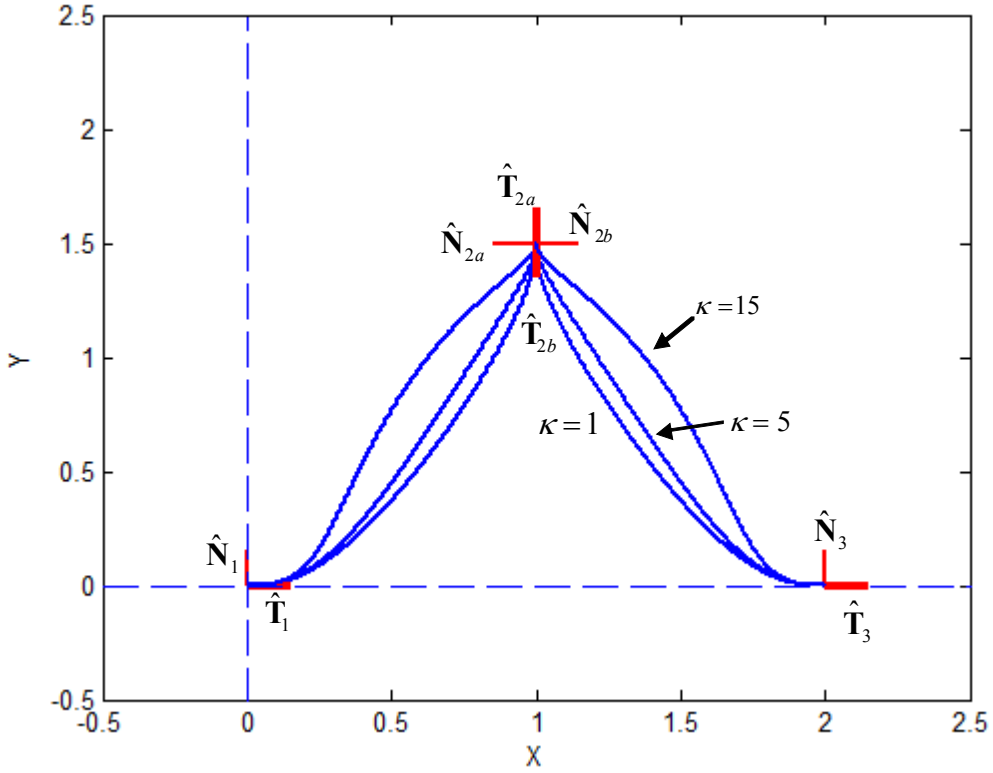


Figure 5.21. Planar Path with a Cusp

A close-up of the cusp point is shown in Figure 5.22. This shows the behavior of the local geometry of the curve around the cusp point for varying values of curvature. Using this method, the effects of curvature on the local geometry of a curve are the same as those described in Sections 4.3.4 and 4.4.2. Thus, the physical meaning of the geometric constraints is retained in a form that allows easy conversion to parametric constraints. While this method does not describe a cusp in the strictest mathematical sense, it does allow for defining the geometric interpretation of a cusp. Table 5.14 and Table 5.15 show the parametric constraints for this cusp example.

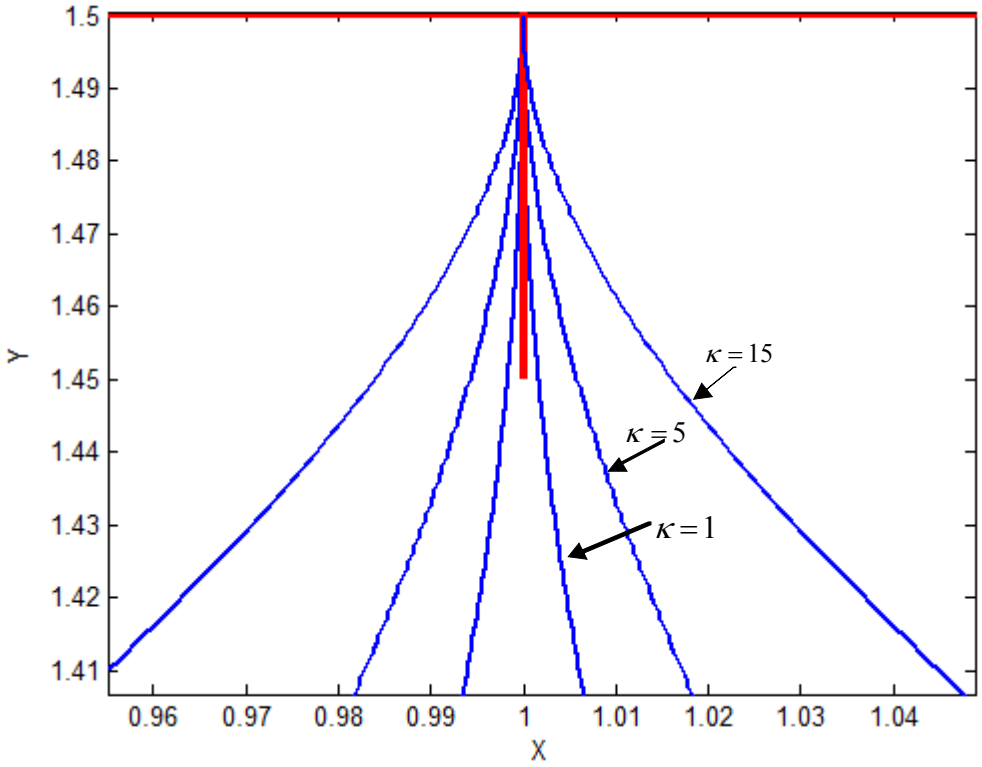


Figure 5.22. Close-up of Cusp Point

	x_{2a}	y_{2a}	$\frac{dx_{2a}}{du}$	$\frac{dy_{2a}}{du}$	$\frac{d^2x_{2a}}{du^2}$	$\frac{d^2y_{2a}}{du^2}$
$\kappa = 1$	1.0	1.5	0.0	1.0	-1.0	0.0
$\kappa = 5$	1.0	1.5	0.0	1.0	-5.0	0.0
$\kappa = 15$	1.0	1.5	0.0	1.0	-15.0	0.0

Table 5.14. Parametric Constraints Approaching Cusp Point

	x_{2b}	y_{2b}	$\frac{dx_{2b}}{du}$	$\frac{dy_{2b}}{du}$	$\frac{d^2x_{2a}}{du^2}$	$\frac{d^2y_{2a}}{du^2}$
$\kappa = 1$	1.0	1.0	0.0	-1.0	-1.0	0.0
$\kappa = 5$	1.0	1.0	0.0	-1.0	-5.0	0.0
$\kappa = 15$	1.0	1.0	0.0	-1.0	-15.0	0.0

Table 5.15. Parametric Constraints Leaving Cusp Point

5.7. SUMMARY

This chapter described a procedure for formulating the geometric properties studied in the last chapter into parametric constraints that can be used to define spatial parametric curves. The main steps behind this process are:

- First, a set of spatial coordinate frames (positions and orientations) is defined. These would most likely come from a simulation or CAD environment and would not have to be defined by an operator.
- The desired geometric properties at each frame are defined (κ, κ', τ, and τ').
- The first order parametric constraint ($\frac{d\mathbf{p}}{du}$) is defined using the unit tangent vector as $\frac{d\mathbf{p}}{du} = \left\| \frac{d\mathbf{p}}{du} \right\| \hat{\mathbf{T}}(\mathrm{u})$
- The second order parametric constraint ($\frac{d^2\mathbf{p}}{du^2}$) is defined using the unit normal vector and desired curvature as $\frac{d^2\mathbf{p}}{du^2} = \left\| \frac{d\mathbf{p}}{du} \right\|^2 \kappa \hat{\mathbf{N}}$
- The third order parametric constraint ($\frac{d^3\mathbf{p}}{du^3}$) is defined using the desired torsion and derivative of curvature as $\begin{bmatrix} \tau \\ \frac{d\kappa}{du}\kappa \left\| \frac{d\mathbf{p}}{du} \right\|^6 \end{bmatrix} = \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \end{bmatrix} \begin{bmatrix} x''' \\ y''' \\ z''' \end{bmatrix}$
- The fourth order parametric constraint ($\frac{d^3\mathbf{p}}{du^3}$) is defined using the derivative of torsion as $\frac{d\tau}{du} \left\| \frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2} \right\|^2 + 2\tau \left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2} \right) \cdot \left(\frac{d\mathbf{p}}{du} \times \frac{d^3\mathbf{p}}{du^3} \right) = a_0 x^{(4)} + b_0 y^{(4)} + c_0 z^{(4)}$
- Once the parametric constraints ($\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^4x}{du^4}, \frac{d^4y}{du^4}, \frac{d^4z}{du^4}$) have been defined, the *x*, *y*, and *z* trajectories are calculated independently using some trajectory planning technique (polynomial, trapezoidal, etc). A comparison of Polynomial vs. Trapezoidal planning is include in Appendix B.

- The individual trajectories are combined together to produce a parametric description of the desired spatial curve $\mathbf{p}(u) = \begin{bmatrix} x(u) & y(u) & z(u) \end{bmatrix}$

This procedure provides a good starting point for defining parametric spatial curves based on geometric constraints. It should be noted that, once the parametric constraints have been defined, any number of curve generation techniques could be used for blending between these constraints. However, there are still many potential areas for future work on this technique. Some possible extensions to this technique include:

- How best to define $u = f(t)$ to define a smooth motion along the defined parametric curve
- A better way of controlling the shape of the curve and its geometric properties between frames
- The application of theses geometric constraints to specific physical tasks

6. CHAPTER SIX

Motion Planner Implementation

In the previous two chapters, a new method of generating the geometry for spatial curves was presented. Chapter Four focused on developing an understanding of the physical meanings of the intrinsic properties of curvature and torsion. Then, Chapter Five presented a technique for converting these properties into parametric constraints that could be used to generate spatial paths. Now, this chapter will explore how these methods can be used in an actual manipulator motion planner. This will be done by integrating this technique into a previously designed motion planning software developed at the Robotics Research Group (RRG). This software package was built using the Operational Software Components for Advanced Robotics (OSCAR) software libraries. The next section will provide a brief background on OSCAR. Then, a description of the overall architecture of the existing Motion Planner will be presented. Then, specific issues involved in integrating this technique into this architecture will be explored such as defining motion along a curve ($u = f(t)$) as well as defining the rotational motion.

6.1. OPERATIONAL SOFTWARE COMPONENTS FOR ADVANCED ROBOTICS

OSCAR is a set of C++ libraries that can be used for modeling and control of serial manipulators. OSCAR contains two main layers: the support layer and the operational layer. The support layer includes low-level modules such as Base (used for error-handling), Math (linear algebra, vector/matrix operations), and Communications (TCP/IP connections). The operational layer includes higher-level modules used for operations such as Forward/Inverse Kinematics, Decision-Making and Motion Planning.

A simple schematic of this framework is shown in Figure 6.1. A much more detailed description of this framework can be found in [23][24].

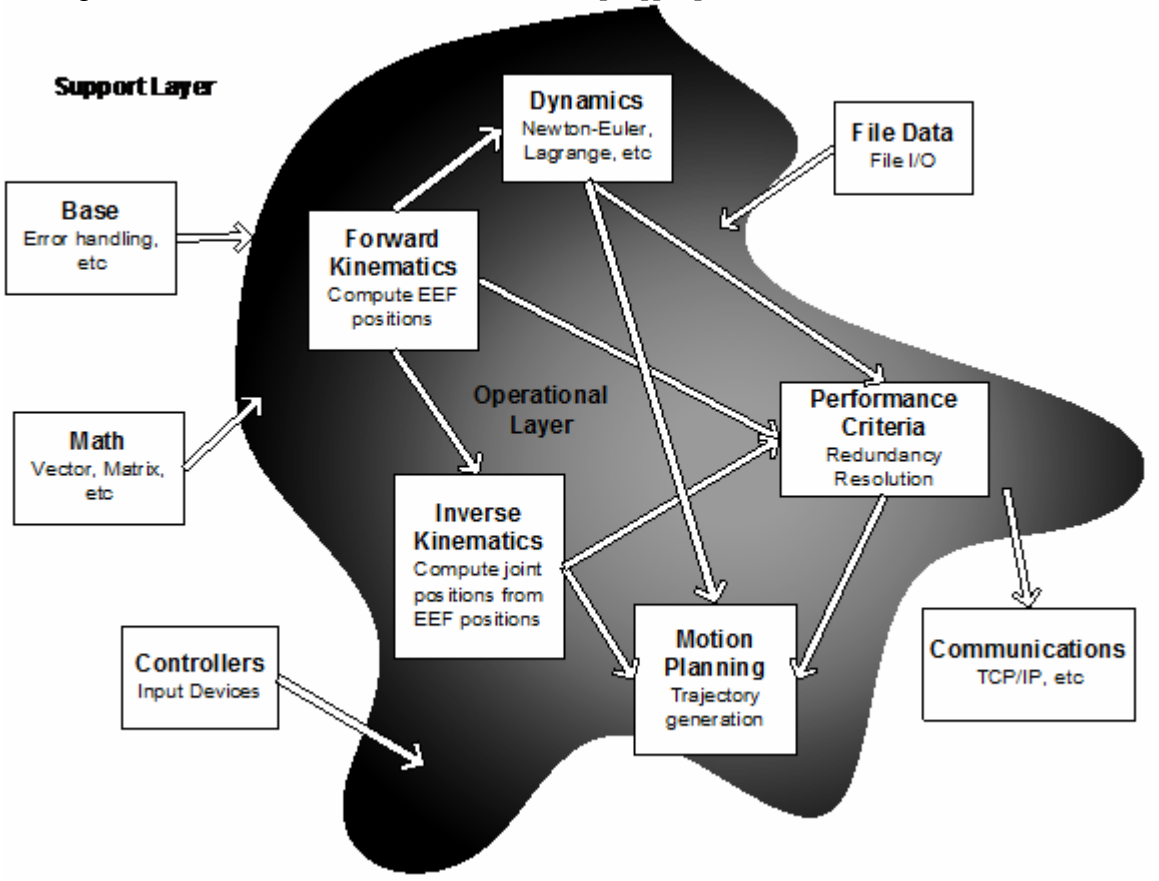


Figure 6.1. OSCAR Architecture Overview

6.2. OSCAR-BASED MOTION PLANNER

The OSCAR-based Motion Planner (MP) is a generalized motion controller designed to package Cartesian and Joint trajectory generation with OSCAR's existing generalized framework for Kinematics and Decision-Making. This includes implementation of Point-to-Point (PTP) motions as well as "jogging" motions for teleoperation. The MP is implemented as a C++ component using OSCAR modules and is robot independent. An overview of the MP framework is shown in Figure 6.2. First, a set of generalized manipulator parameters is provided to create an MP for a specific

robot. Then, a user interface (i.e. API) provides the ability to define joint/Cartesian motions. Once a motion has been defined, the MP will provide joint set points at a specified sample rate that can be sent to a servo interface or simulation. The MP will internally perform Inverse Kinematics when necessary for Cartesian moves as well as checking for joint position and velocity limits. The following sections will briefly describe some of the core functionality of the MP to provide better insight into how the methods developed in this work were integrated.

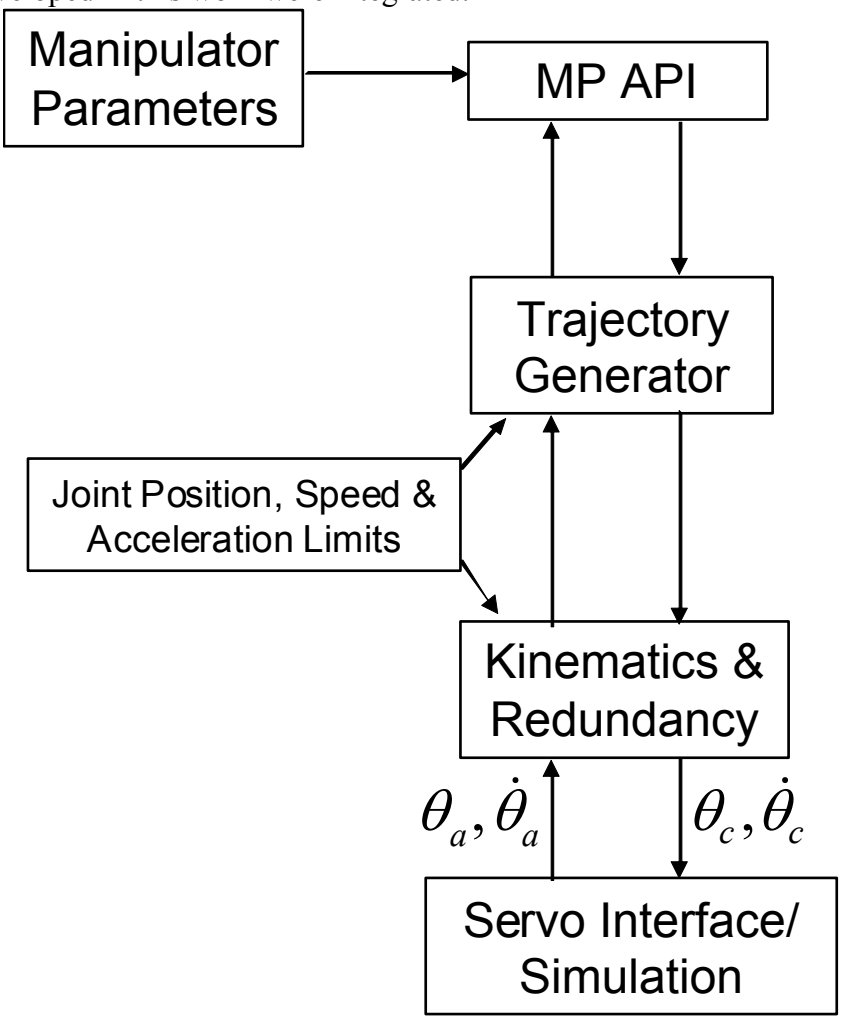


Figure 6.2. Basic Motion Planner Framework

6.2.1. Manipulator Parameters

The manipulator parameters are the robot parameters used to initialize the MP for a specific application. A summary of these parameters is shown in Table 6.1. This table shows that a manipulator-specific MP can be defined with a very simple representation of the system. This allows the MP to be easily applied to a variety of systems.

Manipulator Parameter	**Description**
DH Parameters	Used to describe the spatial geometry of a specific serial manipulator. This information is used by OSCAR for performing the Forward/Inverse Kinematics
Joint Position Limits	Defines the travel limits for each individual joint of the manipulator. These limits are used for error-checking during trajectory calculation/execution.
Joint Velocity Limits	Defines the velocity limits for each individual joint in the manipulator. These limits are used in the generation phase of Joint interpolated moves to ensure the fastest move time based on physical capabilities of the system.
Joint Acceleration Limits	Defines the acceleration limits for each individual joint in the manipulator. These limits are used in the generation phase of Joint interpolated moves to ensure the fastest move time based on physical capabilities of the system..
Cartesian Velocity Limits	Cartesian space velocity limits used for teleoperation.
Cartesian Acceleration Limits	Cartesian space acceleration limits used for teleoperation.

Table 6.1. Basic Manipulator Parameters

6.2.2. Point-to-Point Motions

6.2.2.1. Joint Interpolated

In joint interpolated motions, a desired target joint configuration is provided to the MP. Then, a trajectory for each individual joint is calculated based on the provided joint velocity and acceleration limits to ensure the fastest move time based on system capabilities. This involves an acceleration (i.e. "ramp-up") period, a constant velocity period, and a deceleration (i.e, "ramp-down") period. Then, the longest move time for all joints is determined, and the other trajectories are scaled to complete in the same time period. One example velocity profile is shown in Figure 6.3.

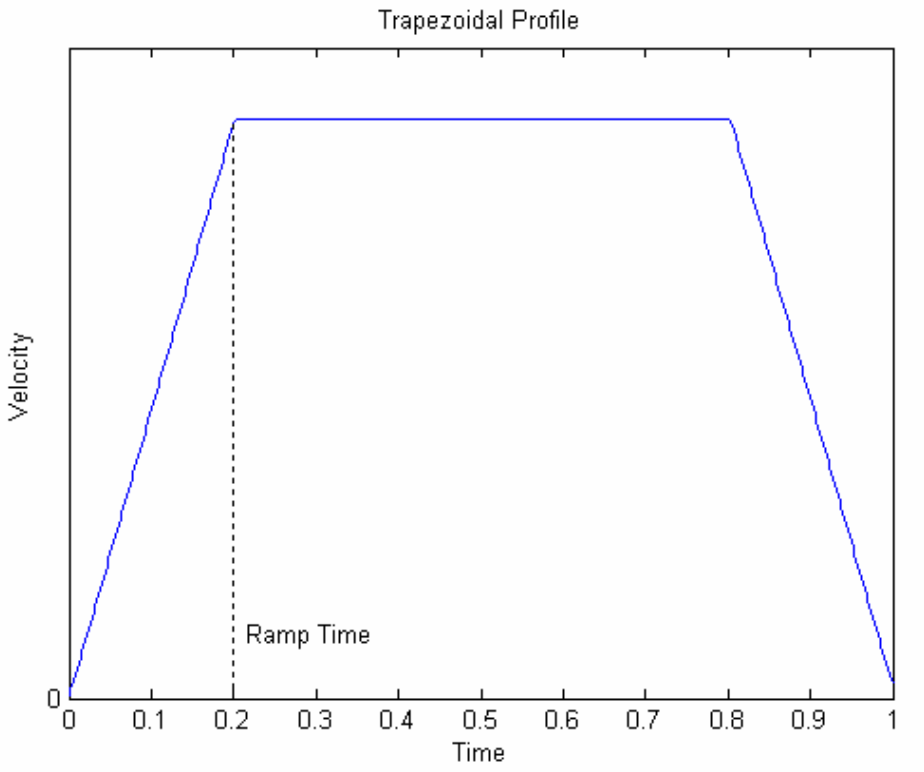


Figure 6.3. Example Velocity Profile

6.2.2.2. Cartesian Interpolated

For Cartesian interpolated moves, a target position and orientation is provided as well as a time to complete the move. The position trajectory is calculated as shown in Equation 6.1 where $\mathbf{p} = [x, y, z]$ and T is the provided move time. As with the joint interpolated motion, the Cartesian motion also involves an acceleration period, a constant velocity period, and a deceleration (i.e, "ramp-down") period. However, in the case of the Cartesian interpolated motions, the velocity profile is based on the distance and move time instead of the velocity/acceleration limits.

$$\mathbf{p}(t) = \mathbf{p}_0 + \frac{t}{T}(\mathbf{p}_1 - \mathbf{p}_0), t \in [0, T] \quad \textbf{6.1}$$

For the rotational motion, a quaternion spherical interpolation (SLERP) technique is used as shown in Equation 6.2 where q_0 and q_1 are quaternions representing the orientation for the beginning and ending points. This type of interpolation provides a smooth angular velocity profile [41].

$$q(t) = \frac{q_0\left(\left(1 - \frac{t}{T}\right)\theta\right) + q_1 \sin(t\theta)}{\sin\theta}, t \in [0, T] \quad \textbf{6.2}$$

6.2.3. Teleoperation

In teleoperation, a set of "delta" values scaled between -1 and 1 is provided to the MP. For joint teleoperation, there will be one delta value for each individual joint. For Cartesian teleoperation, these values represent the *x*, *y*, and *z* translational directions and the θ_x, θ_y, and θ_z Euler angles. The MP will then use its current position/velocity and ramp-up/ramp-down to its set velocity using the provided Cartesian and joint velocity and acceleration limits. For example, if a particular axis is stationary and a delta value of 0.5 is provided, it will ramp-up to 50% of its maximum velocity. The teleoperation functionality of the MP has been tested on actual hardware using a variety of input

devices such as a PC keyboard, a Logitech Magellan Spacemouse, and a Phantom Omni Haptic Device.

6.2.4. Motion Execution

Once a motion has been requested, the MP will provide discrete joint set points at a fixed sampling rate. For a joint interpolated motion, this just means calculating a new joint position based on the motion parameters at a set rate. For Cartesian motions, an end-effector position is calculated at each point and converted into a joint position using Inverse Kinematics. Figure 6.4 shows a simple example of the MP interface. In this example, a Cartesian move to a specified End-Effector position (*finalHand*) is requested for a specific move time (*moveTime*). Then, the MP is polled until the trajectory is complete. In a physical system, the GetJointPosition() method would be placed inside a real-time loop running at a specified sample rate (e.g. 100 hz). However, it is simply placed inside a loop in this example to demonstrate the concept. A full API for the MP is include in Appendix C.

```
moveTime=10.0;
if (!motionPlanner.PlanMove(
      currentJoints,finalHand,CartesianInterpolated,moveTime))
  {
    DisplayError(motionPlanner.GetError());
    return false;
  }

  do{
   if(!motionPlanner.GetJointPosition(currentJoints,jointVel,state)){
      DisplayError(motionPlanner.GetError());
      break;
    }
    SetJoints(currentJoints);
 }while(state != TrajectoryGenerator::TrajectoryComplete);
```

Figure 6.4. Example MP Code

6.2.5. Configuration Parameters

In addition to the core functionality, the MP contains several configuration parameters that can be used to further customize its operation.

Configuration Parameter	Description
RampTime	Changes the % of the velocity to profile to use for the acceleration/deceleration periods for Cartesian interpolated motions (see Figure 6.3).
CoordinateMode	Can be used to toggle between using World and Tool frame coordinates for Cartesian jogging.
TrajectoryShape	Can be used to toggle between Trapezoidal and S-Curve velocity profiles. S-Curve trajectories provide a smoother motion but longer move times.
SpeedScale	Used to slow trajectory execution for debugging purposes. For example, a SpeedScale of 0.5 will cause all trajectories to be executed on 50% speed.

Table 6.2. Example Configuration Parameters

6.2.6. Example Applications

The OSCAR-based MP has been implemented and integrated with several physical systems. A few of the applications for which the MP has been used include:

- **DARPA TraumaPod** – The MP was used for motion control of a Scrub Nurse System (SNS) used for delivering supplies and changing tools inside a surgical workcell. This included the standard operations described in this chapter (Kinematics, Cartesian/Joint motions, etc) as well as integration with Obstacle Avoidance and Collision Detection.

- **Microsoft Robotics Studio** – The MP was integrated with MSRS (Microsoft Robotics Studio) to demonstrate how it could be implemented within a service-based communications architecture.
- **Mobile Manipulator** – The MP was used for motion control for a mobile manipulation system developed with Idaho National Laboratory.
- **LWA3 Teleoperation Demo** – The MP was used to control a 7-DOF LWA3 manipulator using a variety of teleoperation input devices.

6.3. GEOMETRIC-BASED TRAJECTORY GENERATION

Now that the general framework for the OSCAR-based MP has been described, the implementation of the techniques developed as part of this research can be explored. First, a simple task will be defined for demonstration. Then, an exploration on how to define a time-based motion along the geometric trajectory (i.e. how to define $u = f(t)$). will be presented. Then, an introduction to rotational motion will be provided.

6.3.1. Task Description

To examine the specific details of implementing the path generation techniques developed in this work, a simple example task was chosen. This task involves passing through a waypoint with a specified curvature in route to a final position (Figure 6.5). Located at the waypoint position is a sphere with a radius of 0.05 meters. Thus, a curvature value of $\kappa = 20$ ($\kappa = \frac{1}{\rho}$) will make the local shape of the curve at the waypoint identical to the sphere. While this is a very basic example, it provides a good geometric visualization of the generated spatial curve. The same methods developed to interpolate this curve can also be applied to more complex examples.

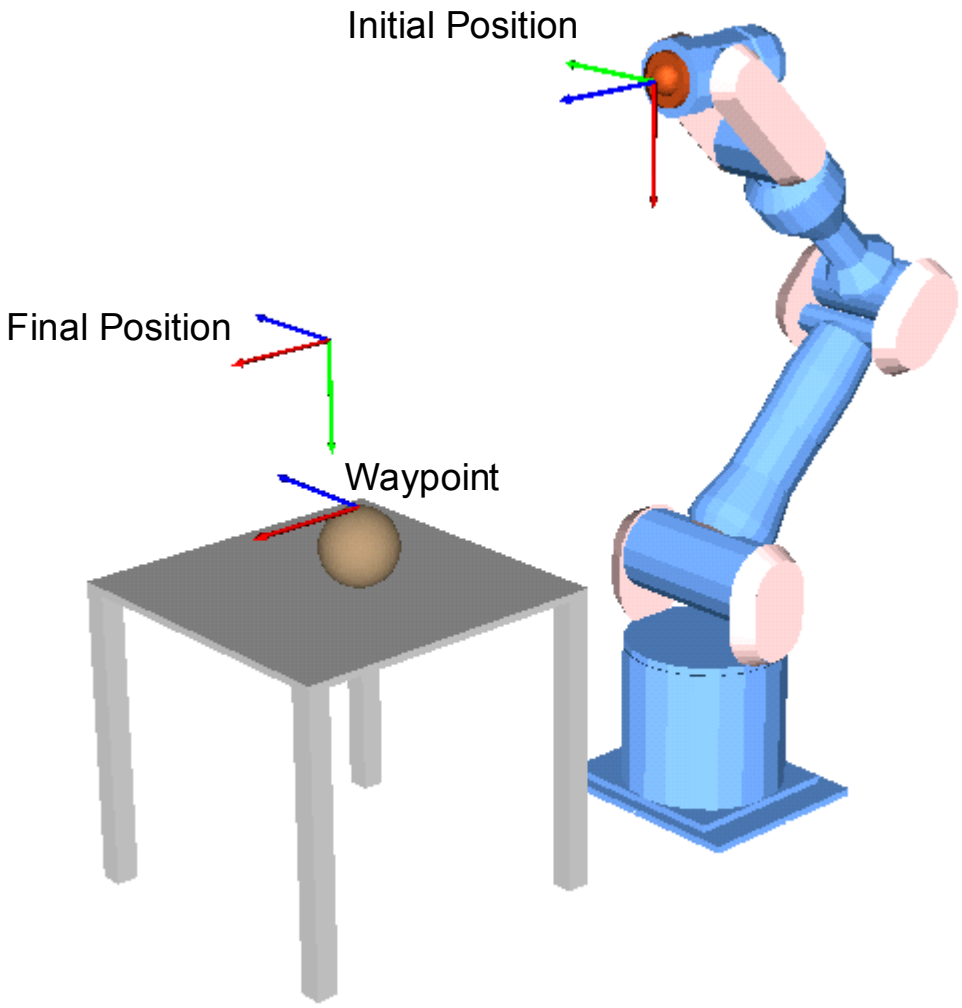


Figure 6.5. Sample Task Example

6.3.2. Translational Motion Along a Curve

The first step in this implementation is how to describe the translational motion along a curve (i.e. $u = f(t)$). As mentioned earlier in the chapter, the Motion Planner works by providing joint set positions at a specified sample rate. Thus, the task here is to figure out how to sample points on the curve to provide a desirable motion. This comes down to determining how to increment the geometric parameter u. A basic schematic of

the process that will need to be computed at each sampling point[9] is shown in Figure 6.6. The following sections will go into more detail of how this process is realized.

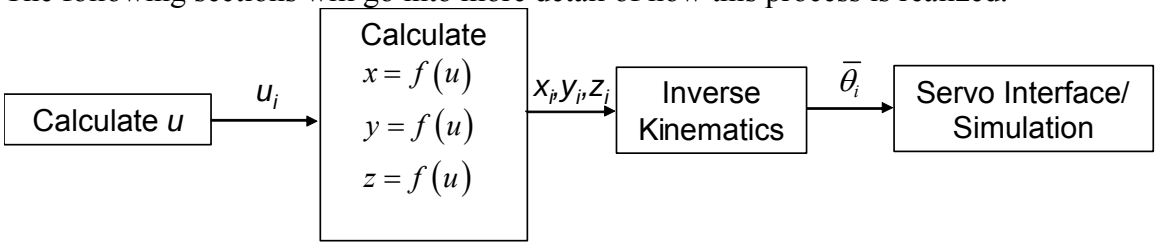


Figure 6.6. Basic Schematic of MP Curve Interpolator

6.3.2.1. Effect of Tangent Scale

In Section 5.2, it was shown that the first-order parametric properties could be set by scaling the unit tangent vector (i.e. $\frac{d\mathbf{p}}{du} = tangScale * \hat{\mathbf{T}}$ where *tangScale* is some scalar value). Because Chapter 5 mainly focused on the higher-order properties, this parameter was not fully explored. Figure 6.7 shows the affects of this parameter as applied to the current task. This shows that this parameter has a large affect on the overall shape of the curve while still allowing the geometric constraints to be met at the end points. Basically, as the value of *tangScale* increases the curve tends to develop large overshoots. For lower values, the curve will become more "taut". The exact affect of this parameter on a curve will be dependant on both the scale (e.g. meters vs. millimeters) as well as the values of the higher-order properties. Thus, first the local geometric properties can be defined, and then this parameter can be tweaked to find a desirable overall curve shape.

[9] This schematic only shows the translational component of the position and not the orientational. For now, the orientation can be thought of as a constant along the path.

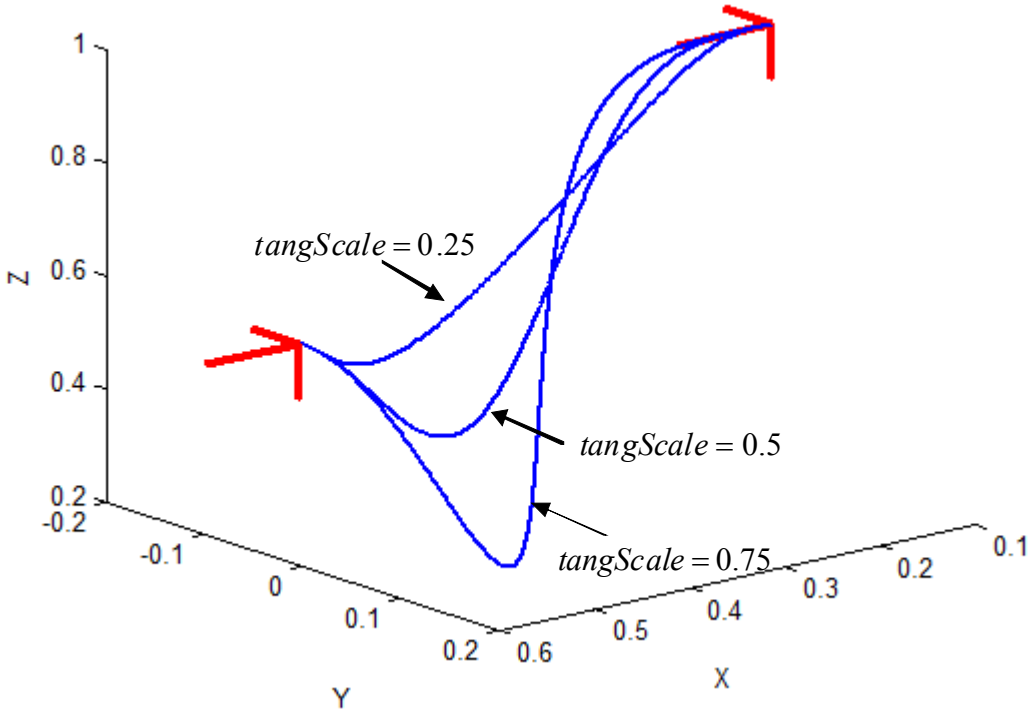


Figure 6.7. Effect of varying Tangent Scale

Using a value of *tangScale*=0.3, the geometry of the path can now be defined. The resulting curve is shown in Figure 6.8. This figure shows that the curve passes through the waypoint while locally tracing the surface of the sphere as expected. Now that the geometry has been defined, the next sections will describe how to define a motion along this geometry.

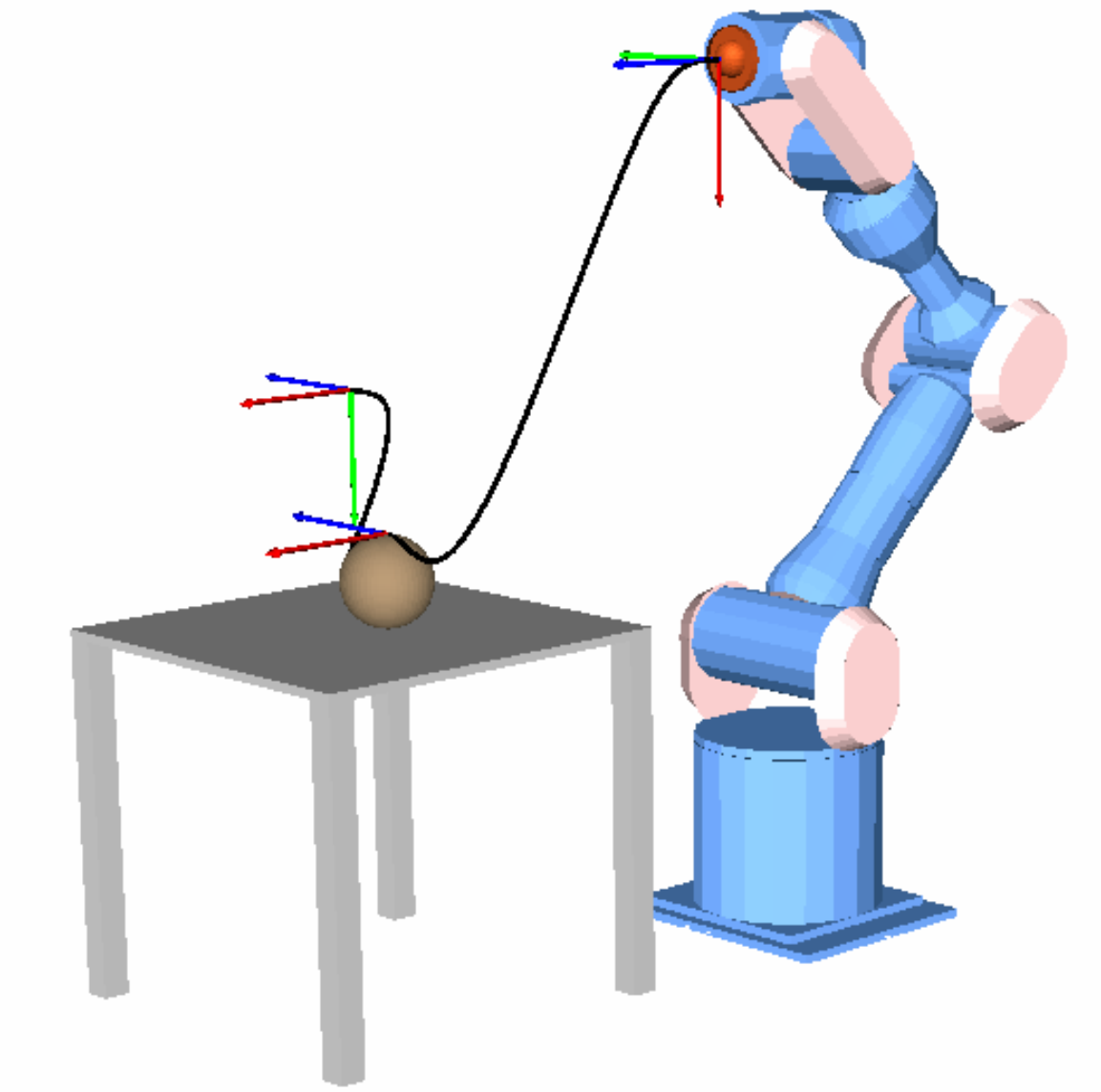


Figure 6.8. Resulting Spatial Curve

6.3.2.2. Linear Parametric Interpolation

The simplest method of interpolating the geometric parameter u would be to linearly interpolate it with respect to the time. This is shown in Equation 6.3 where T is the total time to complete the move. From this equation, it is easy to see that $u(0) = u_i$ and $u(T) = u_f$. This simple method was implemented into the MP and the output Cartesian velocity along the path was measured by calculating the distance between

consecutive set points provided by the MP and dividing by the sample time. The results of the simulation are shown in Figure 6.9.

$$u(t) = u_i + \frac{t}{\mathrm{T}}\left(u_f - u_i\right), t \in [0, \mathrm{T}] \qquad \textbf{6.3}$$

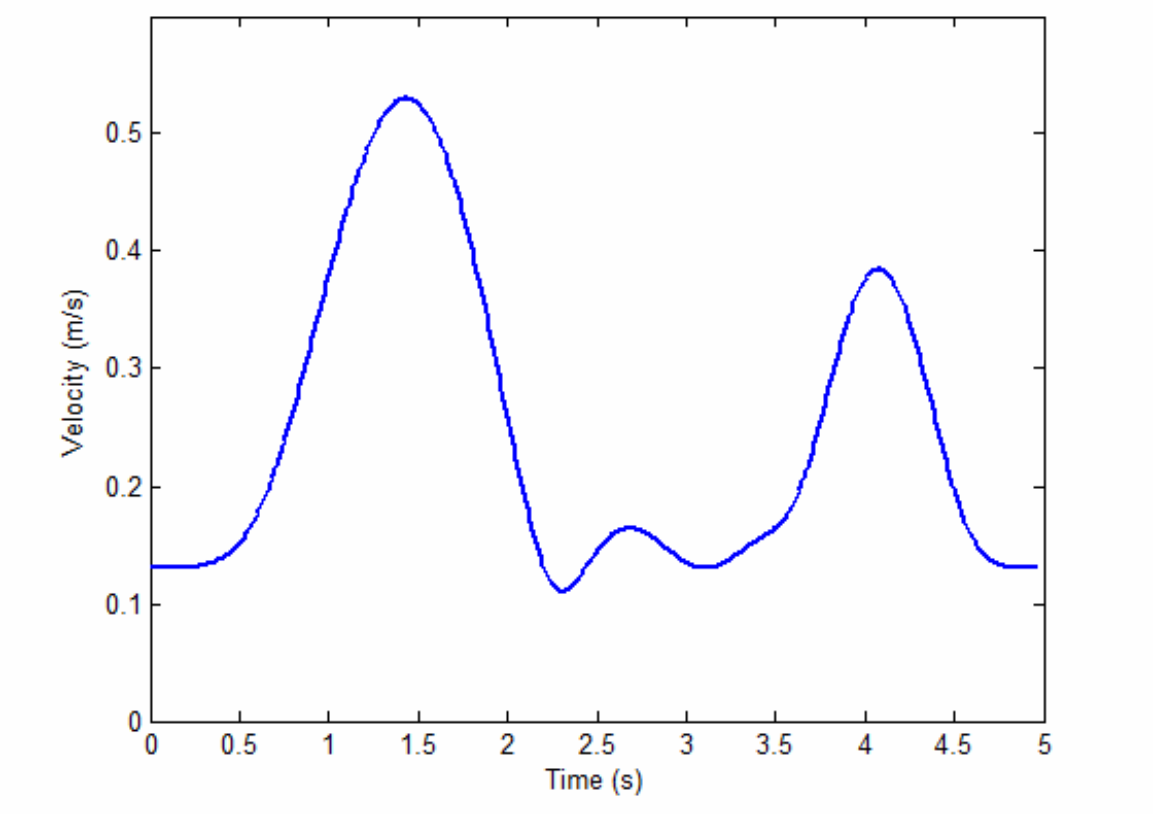


Figure 6.9. Velocity Profile for Linear *u* Interpolation

The above velocity profile shows that this leads to a smooth but uncontrolled velocity in the middle of the trajectory. However, the trajectory also has a finite initial/final velocity which will lead to large accelerations and undesirable motions.

6.3.2.3. Smooth Parametric Interpolation

A simple extension to the linear parametric interpolation is to use a smooth function to interpolate the parameter. One simple example of this is shown in Equation 6.4. This function will satisfy the conditions $p(0) = 0$ and $p(T) = 1$ with both the initial

and final velocity and acceleration beginning zero. This will ensure a smooth starting and ending motion for the trajectory.

$$p(t) = 10\left(\frac{t}{T}\right)^3 - 15\left(\frac{t}{T}\right)^4 + 6\left(\frac{t}{T}\right)^5, t \in [0, T] \quad \textbf{6.4}$$

Then, the geometric parameter u can be defined as a function of time as shown in Equation 6.5 where $u(0) = u_i$ and $u(T) = u_f$. Once again, the resulting Cartesian velocity is calculated and plotted in Figure 6.10. This plot shows that the initial and final velocities are indeed zero now (providing a smooth start and finish), but the interior velocity profile (peak magnitude) is still not controlled. Thus, a more detailed examination between the geometry and motion of the curve is needed. This is presented in the following section.

$$u(t) = u_i + p(t)\left(u_f - u_i\right), t \in [0, T] \quad \textbf{6.5}$$

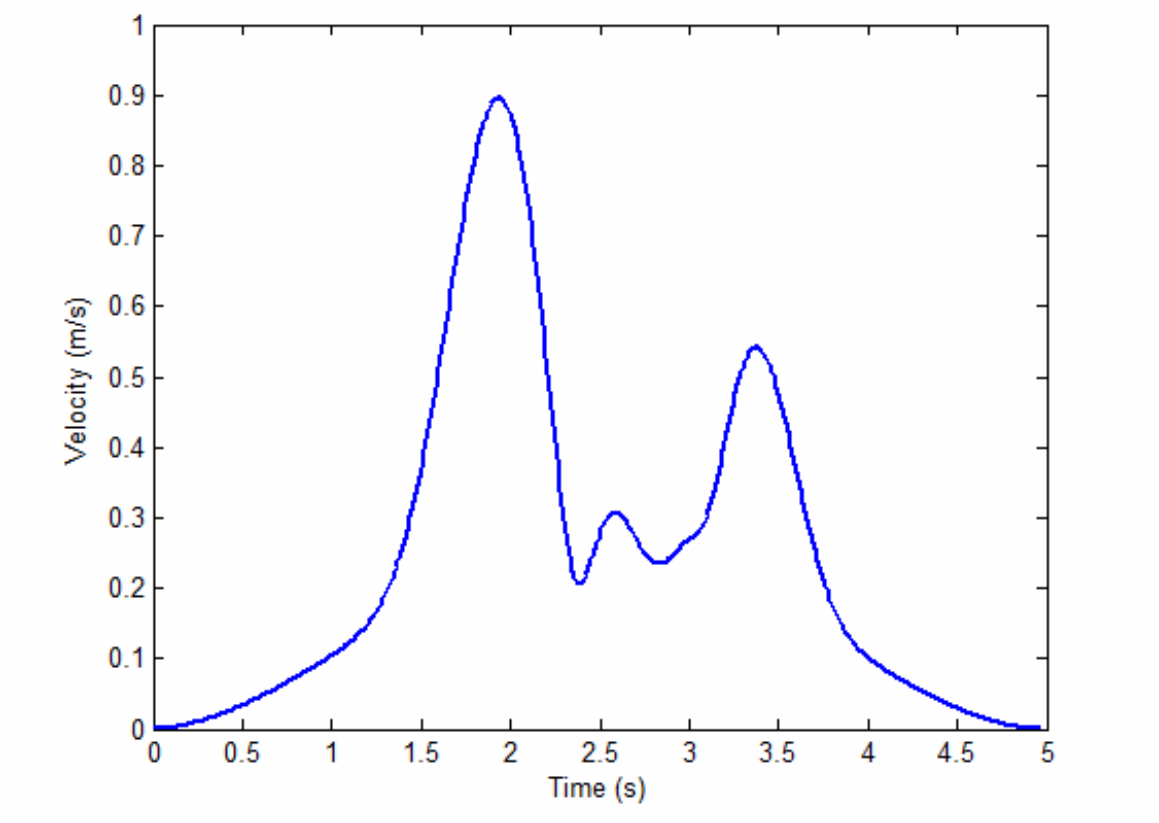


Figure 6.10. Velocity Profile for Smooth *u* Interpolation

6.3.2.4. Velocity Approximation Formulation

In Section 2.3.2, an introduction to motion along a curve parameterized by arc length, *s*, was presented. In curves parameterized by arc length, developing relationships between geometry and motion is simple as the value $\dot{s} = \frac{ds}{dt}$ is the Cartesian speed (i.e. magnitude of the velocity). However, in this case, the curve is defined not by arc length but by a geometric parameter *u*. In this section, a method will be described to move along a parametric curve with a prescribed velocity profile. This method is based off a real-time interpolator developed for CNC machines by Zhang and Greenway [63].

The first step is to take a Taylor expansion of the function $u(t)$ as shown in Equation 6.6 where Δt is the time between sampling periods (e.g. 0.01 seconds for 100 hz). Using this formulation, the next value of *u* can be calculated at each sampling period by using the current value of *u* as well as the higher-order derivatives of *u* with respect to time. Thus, these higher-order derivatives of *u* with respect to time must be formulated. This will first be done for a first-order approximation and then expanded to a second-order approximation.

$$u_{i+1} = u_i + \Delta t \frac{du_i}{dt} + \frac{\Delta t^2}{2}\frac{d^2u_i}{dt^2} + \cdots + \text{H.O.T} \qquad \textbf{6.6}$$

For a first-order approximation, the value of $\frac{du}{dt}$ must be found. First, the relationship shown in Equation 6.7 is developed. This relates the physical motion along the curve to the geometric parameter.

$$v = \dot{s} = \frac{ds}{dt} = \frac{ds}{du}\frac{du}{dt} \qquad \textbf{6.7}$$

Now, we can substitute in the relationship $\left\| \frac{d\mathbf{p}}{du} \right\| = \frac{ds}{du}$. This yields the formulation for $\frac{du}{dt}$ shown in Equation 6.8.

$$\frac{du}{dt} = \frac{v(t)}{\left\| \frac{d\mathbf{p}}{du} \right\|} \tag{6.8}$$

This shows that the value for $\frac{du}{dt}$ can be found at any point along the curve given the current desired velocity. Substituting this into Equation 6.6 will yield the relationship shown in Equation 6.9. Thus, at each sample time, a new value of *u* can be calculated.

$$u_{i+1} = u_i + \frac{v(t)\Delta t}{\left\| \frac{d\mathbf{p}_i}{du} \right\|} \tag{6.9}$$

Now that a method for determining *u* has been developed, the velocity profile *v(t)* must be defined. For the purposes of testing this formulation, a simple trapezoidal velocity profile will be defined as shown in Figure 6.11.

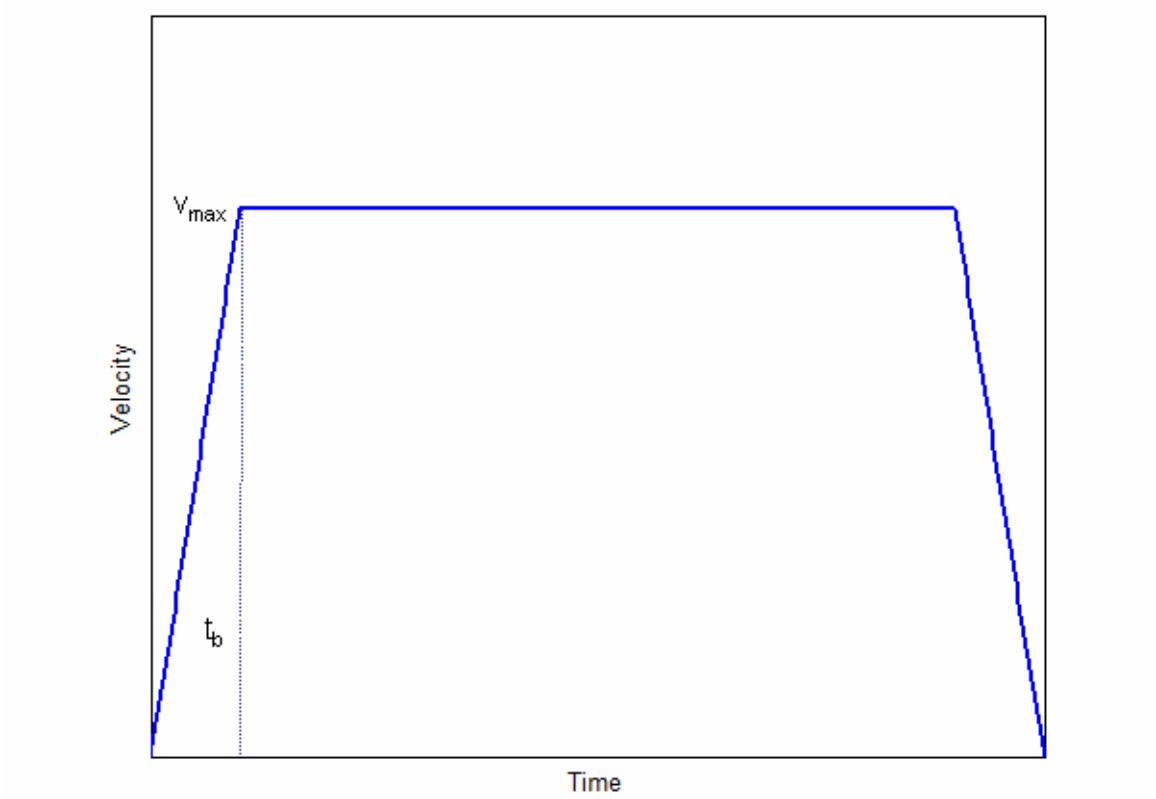


Figure 6.11. Example Velocity Profile

Before we can calculate the function *v(t)*, the total distance travelled along the curve must be calculated. This arc length can be found using Equation 6.10. A simple way of computing this integral is to sample *u* across its interval and sum up the distances between consecutive points (*x,y,z*).

$$s = \int_{u_i}^{u_f} \sqrt{\frac{d\mathbf{p}}{du} \cdot \frac{d\mathbf{p}}{du}} du = \int_{u_i}^{u_f} \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2} du \qquad \textbf{6.10}$$

Now that the total travel distance has been computed, the coasting velocity and ramp-up acceleration can be calculated using Equation 6.11 and 6.12.

$$v_{\max} = \frac{s}{T - t_b} \qquad \textbf{6.11}$$

$$a_{\max} = \frac{v_{\max}}{t_b} \qquad \textbf{6.12}$$

The entire velocity function can be described as shown in Equation 6.13[10]. Now, this method for defining motion along a curve can be tested as before. The resulting velocity profile is shown in Figure 6.12. From this plot, the first-order approximation does a decent job of tracking the velocity profile with the exception of one small area where the curvature is large. As mentioned in [63], the first-order approximation is only reliable in curves with small values of curvature. However, since the technique described in this work depends on defining curvatures that may in some cases be large, a higher-order approximation is required.

$$v(t) = \begin{Bmatrix} a_{\max} t, & t < t_b \\ v_{\max}, & t_b < t < T - t_b \\ v_{\max} - a_{\max} t, & T - t_b < t < T \end{Bmatrix} \qquad \textbf{6.13}$$

[10] A similar formulation could be created where desired velocity/acceleration are the inputs and T/t_b are calculated values.

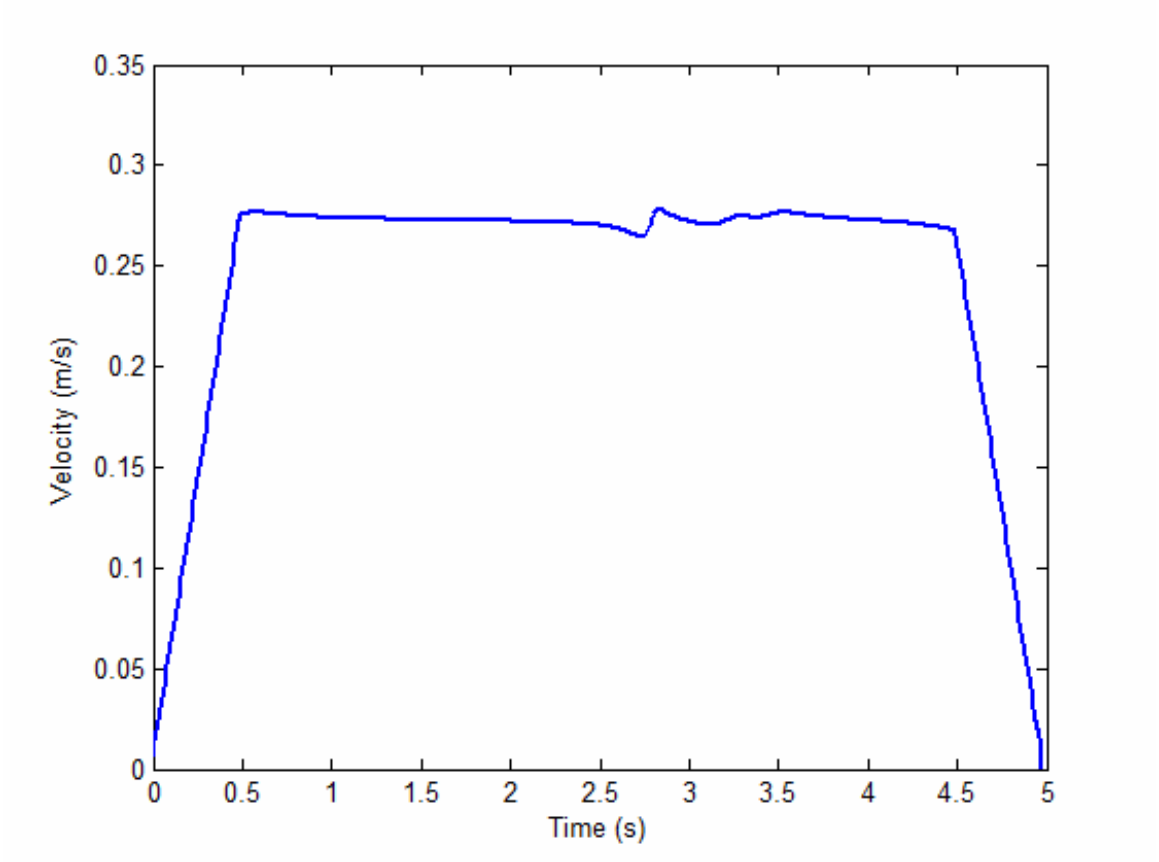


Figure 6.12. First-Order Velocity Approximation

To define a second-order approximation, the value of $\frac{d^2u}{dt^2}$ must be formulated. This can be done by simply taking the derivative of $\frac{du}{dt}$ as shown in Equation 6.14. Then the overall method of interpolating the geometric parameter *u* is shown in Equation 6.15. The resulting velocity profile is shown in Figure 6.13. This shows that a second-order approximation tracks the desired velocity profile almost exactly.

$$\frac{d^2u}{dt^2} = \frac{d}{dt}\left(\frac{v(t)}{\left\|\frac{d\mathbf{p}}{du}\right\|}\right) = \frac{a(t)}{\left\|\frac{d\mathbf{p}}{du}\right\|} - \frac{v(t)^2\left(\frac{d\mathbf{p}}{du}\cdot\frac{d^2\mathbf{p}}{du^2}\right)}{\left\|\frac{d\mathbf{p}}{du}\right\|^4} \qquad \textbf{6.14}$$

$$u_{i+1} = u_i + \Delta t\left(\frac{v(t)}{\left\|\frac{d\mathbf{p}_i}{du}\right\|}\right) + \frac{\Delta t^2}{2}\left(\frac{a(t)}{\left\|\frac{d\mathbf{p}}{du}\right\|} - \frac{v(t)^2\left(\frac{d\mathbf{p}}{du}\cdot\frac{d^2\mathbf{p}}{du^2}\right)}{\left\|\frac{d\mathbf{p}}{du}\right\|^4}\right) \quad \textbf{6.15}$$

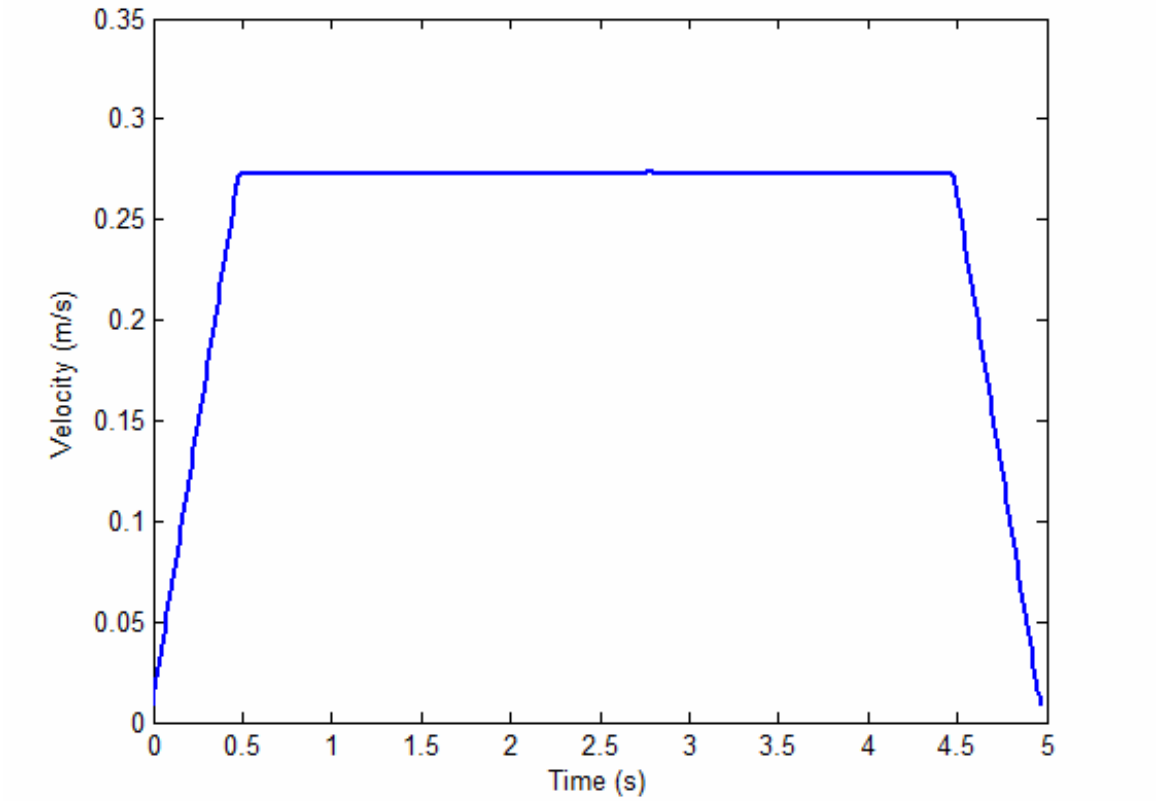


Figure 6.13. Second-Order Velocity Approximation

6.3.2.5. Special Cases

The method described in the previous section assumes that a nonzero velocity profile is desired along the entire manipulator trajectory. This is a decent assumption for most cases as the curve generation method developed in this work is in general smooth over the whole trajectory. However, in certain special cases, a discontinuity is purposefully inserted into the trajectory. For example, consider the trajectory with a cusp shown in Figure 6.14. In this example, the manipulator is commanded to move into a tray dispenser to pick up a tray and then on to a final position.

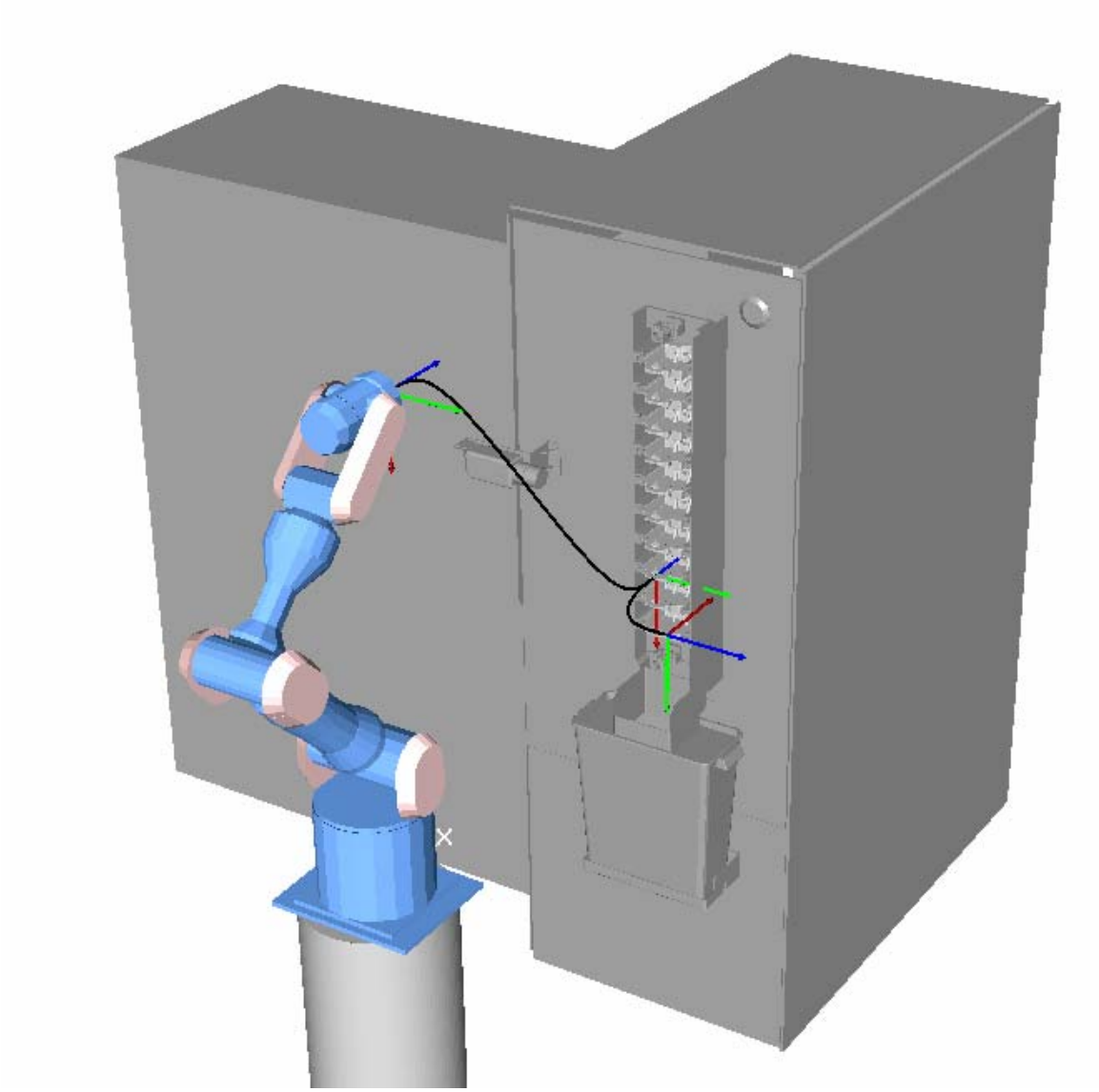


Figure 6.14. Example Trajectory with a Cusp

In the above trajectory, it is easy to see that the curve switches directions at the cusp point. Thus, if the manipulator is simply commanded to traverse the curve at a constant speed, the motion of the trajectory will instantaneously switch directions at this point causing high shocks in the physical system. The simplest method of getting around this is to define a different velocity profile for each portion of the curve with the velocity being zero at the actual cusp point. An example of this is shown in Figure 6.15. It should

be noted that depending on the application, a zero velocity could be desired even at positions where the parametric description remains smooth.

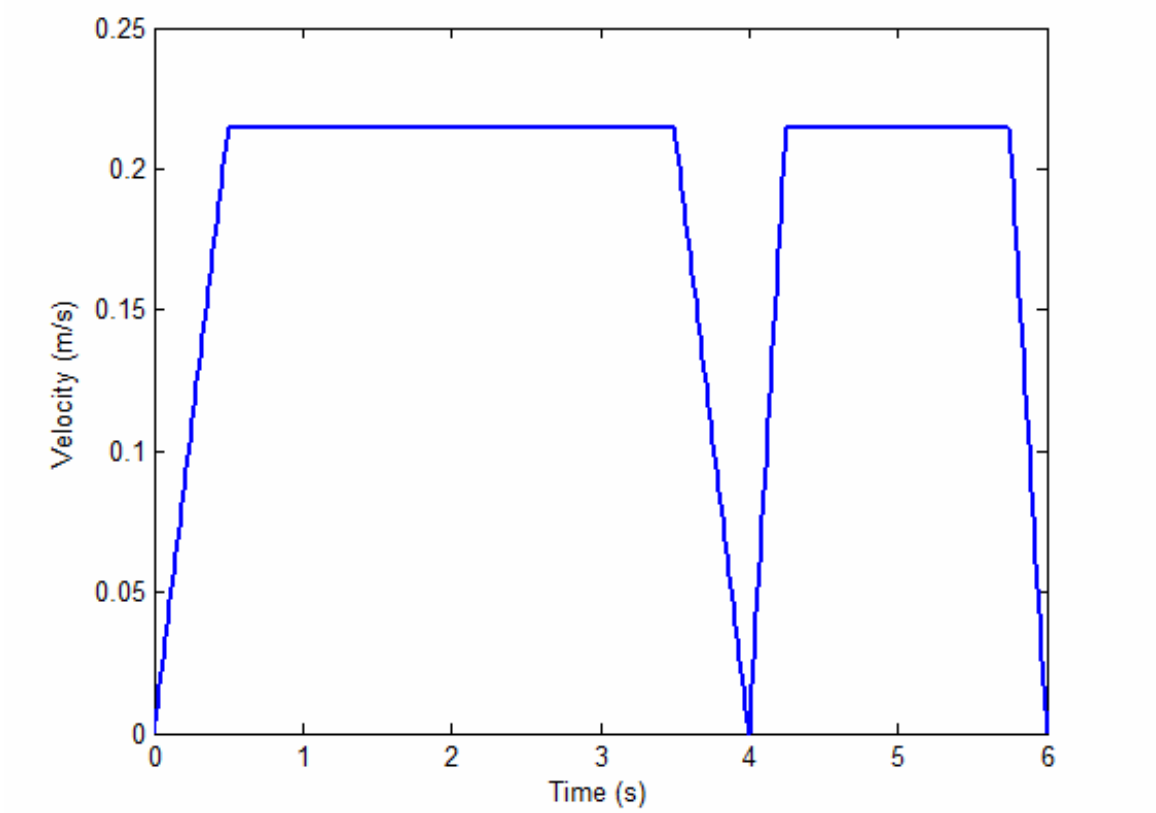


Figure 6.15. Example Velocity Profile

6.3.2.6. Conclusions

This section explored methods of generating a time-based motion along a parametrically defined spatial curve. This is necessary for implementing the proposed methods on an actual physical system. First, it was shown that a simple approach of defining $u=t$ would not lead to desirable or controllable output velocities. Then, a method that used the geometric properties of the curve in combination with a defined velocity profile was presented. This was first done for a first-order approximation that tracked the velocity profile nicely in areas of low curvature. This was then expanded to a second-order approximation that leads to a good, controllable velocity along the geometric path.

This technique is applicable to any parametrically defined spatial curve as long as the first and second order properties are available. Thus, if different methods of defining the blending between end constraints are developed in the future, this same process remains useful.

6.3.3. Rotational Motion

This research has mainly been focused on the generation of spatial curves for the purposes of translational path planning. However, to fully define the motion of a robot manipulator, the orientation of the end-effector must be defined as well as the position. In this section, an introductory look at rotational motion planning will be presented.

6.3.3.1. Orientation-to-Orientation Interpolation

The simplest form of rotational motion planning is defining an interpolation between an initial and final frame. For example, for the simple task described earlier in the chapter, the orientations may be defined as shown in Figure 6.16. In this rotational description, the manipulator is commanded to pass through the waypoint with an orientation normal to the sphere and then return to its original orientation. This section will explore some of the basic ways to describe this motion.

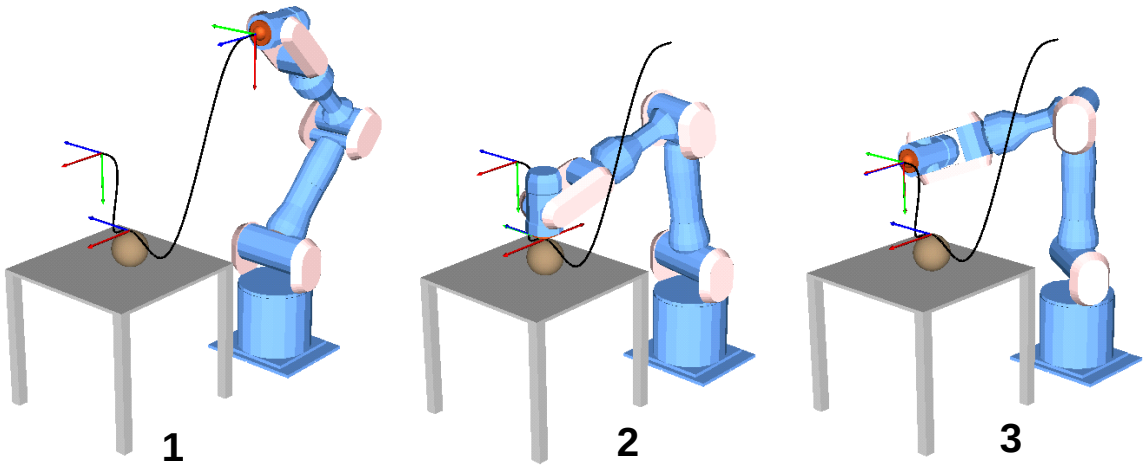


Figure 6.16. Sample Orientation Descriptions

One of the most commonly used descriptions for an orientation is Euler Angles (introduced in Section 1.1.2.2). With Euler Angles, each orientation (i.e. rotation matrix) is converted into a set of three angles $[\alpha, \beta, \gamma]$ that represent three consequence rotations to reach the final desired orientation. The interpolation between two sets of Euler Angles can then be performed as in Equation 6.16 where the Euler Angles can be converted into a rotation matrix at each instance of *t*. While this provides a simple way of describing the rotational motion, Euler Angles suffer from many problems. First, they are prone to singularities such as gimbal lock. Second, they do not necessarily provide a controllable angular velocity along their trajectory.

$$\left.\begin{aligned} \alpha(t) &= \alpha_i + \frac{t}{T}(\alpha_f - \alpha_i) \\ \beta(t) &= \beta_i + \frac{t}{T}(\beta_f - \beta_i) \\ \gamma(t) &= \gamma_i + \frac{t}{T}(\gamma_f - \gamma_i) \end{aligned}\right\}, t \in [0, T] \qquad \textbf{6.16}$$

For this reason, rotational motions are often described using Equivalent Axis formulations. The basic Equivalent Axis formulation (as first described in Section 1.1.2.3) has the form $R(\hat{\mathbf{n}}, \theta)$ where $\hat{\mathbf{n}}$ describes a spatial vector and θ describes the angle with which to rotate around that axis. Now in the simple case where a rotational motion from the rotational identity matrix to some desired frame needs to be generated, the rotation along the trajectory can simply be defined as $R\left(\hat{\mathbf{n}}, \frac{t}{T}\theta\right), t \in [0, T]$. Thus, a rotation matrix can be calculated at each time instance *t*. Because this representation is simply a rotation about a fixed axis, the quantity $\dot{\theta}$ can be controlled providing a much better physical understanding of the angular motion. Now, suppose this method needs to be used to interpolate between two arbitrary frames *A* and *B* as shown in Figure 6.17.

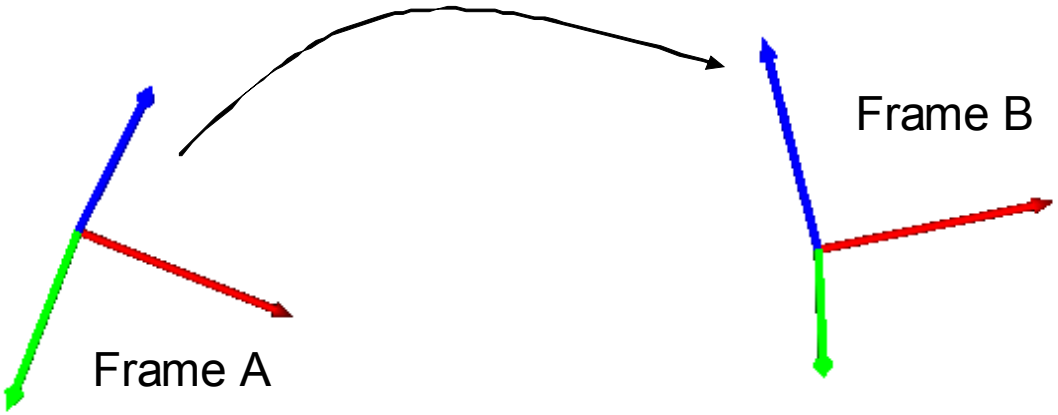


Figure 6.17. Example Rotational Interpolation

First, we calculate the rotation matrix $R = A^T B$ which basically describes the relative rotation between A and B. Then, Equivalent Axis and Angle for this rotation are calculated. Then, this rotation is transformed back into Frame A and the final motion can be defined as $AR\left(\hat{\mathbf{n}}, \frac{t}{T}\theta\right)$. Thus, a smooth rotation with a controllable angular velocity is defined between two arbitrary frames.

Another representation for orientations that is similar to Equivalent Axis is quaternions (first introduced in Section 1.1.2.4). Quaternions are defined as a four-dimensional vector representing an axis and rotation in space. They can be defined from an Equivalent Axis formulation as shown in Equation 6.17 [21].

$$\begin{aligned} q_0 &= \cos\left(\frac{\theta}{2}\right) \\ q_1 &= n_1 \sin\left(\frac{\theta}{2}\right) \\ q_2 &= n_2 \sin\left(\frac{\theta}{2}\right) \\ q_3 &= n_3 \sin\left(\frac{\theta}{2}\right) \end{aligned} \qquad \textbf{6.17}$$

Mathematically, quaternions can be thought of as positions on a 4-dimensional unit hypersphere and interpolation between quaternions can be thought of as curves running along the surface of these spheres. While these interpretations can be difficult to visualize, they provide a powerful way of defining orientation-based curves. For example, consider the simple Spherical Linear Interpolation (SLERP) shown in Equation 6.18. This represents a minimum arc length curve between two points on the sphere.

$$q_{12}(t) = q_1 \frac{\sin\left((1-t)\theta_{12}\right)}{\sin\theta_{12}} + q_2 \frac{\sin t\theta_{12}}{\sin\theta_{12}}, t \in [0,1] \qquad \textbf{6.18}$$

Hanson [21] further shows how these relationships can be nested to generate orientation "curves" with multiple control points. An example of this is shown in Equation 6.19 for a curve through three quaternions where $\theta(t) = \cos^{-1}\left(q_{12}(t) \cdot q_{23}(t)\right)$. While quaternions are mathematically complex and somewhat difficult to visualize, they remain an important area of research due to their widespread use in animation, graphics, and robotics.

$$q_{123}(t) = q_{12}(t)\frac{\sin\left((1-t)\theta(t)\right)}{\sin\theta(t)} + q_{23}(t)\frac{\sin t\theta(t)}{\sin\theta(t)}, t \in [0,1] \qquad \textbf{6.19}$$

6.3.3.2. Geometric or Task-based Rotational Motions

One popular method of defining rotational motion is to tie it to the geometry of a curve or surface. For a curve, this can be done by defining the orientation of the end-effector relative to the Frenet Frame [3][57][58]. For example, consider the rotation matrix shown in Equation 6.20[11]. This would align the *x*, *y*, and *z* axes of the manipulator end-effector frame (i.e. tool frame) with the tangent, normal, and bi-normal vectors of the Frenet Frame, respectively. Normally, the *z* axis of the manipulator end-effector is aligned with the tool axis. So, it is often useful to describe the orientation as in Equation

[11] This is a valid rotation matrix, because the unit tangent, normal, and bi-normal vectors are orthogonal.

6.21 to align the tool axis with the tangent vector. The first column of this rotation matrix is negated to preserve a right-handed coordinate system.

$$R(u) = \left[\hat{\mathbf{T}} \;\; \hat{\mathbf{N}} \;\; \hat{\mathbf{B}}\right] \tag{6.20}$$

$$R(u) = \left[-\hat{\mathbf{B}} \;\; \hat{\mathbf{N}} \;\; \hat{\mathbf{T}}\right] \tag{6.21}$$

There are several advantages to defining the rotational motion in this fashion. First, for certain tasks, this is a natural way to define the orientation. Second, the Frenet-Serret formulas (shown in Equation 6.22 and first introduced in Section 2.3.1.4) provide an intuitive (physical meaning) method for defining the rotational motion as the translational path is transversed.

$$\begin{bmatrix} \frac{d\hat{\mathbf{T}}}{ds} \\ \frac{d\hat{\mathbf{N}}}{ds} \\ \frac{d\hat{\mathbf{B}}}{ds} \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}} \\ \hat{\mathbf{N}} \\ \hat{\mathbf{B}} \end{bmatrix} \tag{6.22}$$

This shows that the same geometric constraints that were used to define the translational path can be useful in defining the rotational motion as well. Hanson [19][21] showed that a similar formulation can be created using quaternions as shown in Equation 6.23. Thus, the movement of the quaternion (aligned with the Frenet Frame) along a parametric curve can be defined in terms of the intrinsic properties of curvature and torsion.

$$\begin{bmatrix} \frac{dq_0}{ds} \\ \frac{dq_1}{ds} \\ \frac{dq_2}{ds} \\ \frac{dq_3}{ds} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\tau & 0 & -\kappa \\ \tau & 0 & \kappa & 0 \\ 0 & -\kappa & 0 & \tau \\ \kappa & 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{6.23}$$

Another important result related to the Frenet-Serret formulas is the Darboux vector: $\mathbf{D} = \tau\hat{\mathbf{T}} + \kappa\hat{\mathbf{B}}$ [26]. This is a rotation vector defining the instantaneous axis and magnitude of rotational motion (analogous to an instantaneous screw motion). The axis defined by **D** is simply the rotational axis and the magnitude of this vector is the angular velocity ω. From this equation, if $\kappa = 0$ and $\tau = 0$, this will lead to no rotational motion as the curve is simply a straight line. Similarly, if $\tau = 0$ and curvature has some positive non-zero value, the frame will simply rotate about the bi-normal axis (i.e. stay in the osculating plane). Thus, the instantaneous rotation around the curve can be calculated at any point from the local curvature and torsion.

However, there are some disadvantages to defining a global motion based on the Frenet Frame with the current path planning method. Because the properties of curvature and torsion are only being constrained at key frames, the Frenet Frame is free to spin around the path during motions. This leads to joint position and/or velocity limits being violated in a serial manipulator. However, it may be possible to use these geometric properties to define the local rotational motion about a specific frame.

An analogous method to the Frenet Frame description can be used to define the rotational motion along a parametric surface. For example, consider the paraboloid (i.e. parabolic surface) described by Equation 6.24.

$$\begin{aligned} x(u,v) &= u \\ y(u,v) &= v \\ z(u,v) &= h\left(u^2 + v^2\right) \end{aligned} \qquad \textbf{6.24}$$

To define a rotation relative to the surface, we first calculate the principal tangent directions as shown in Equation 6.25 and 6.26. These are simply calculated by taking the partial derivatives of the parametric description with respect to the two independent parameters.

$$\mathbf{T}_1 = \left[\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right] = [1,\ 0,\ 2uh] \qquad \textbf{6.25}$$

$$\mathbf{T}_2 = \left[\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}\right] = [0,\ 1,\ 2vh] \qquad \textbf{6.26}$$

Then, the normal to the surface can be computed by taking the cross product of the two principal tangent directions. This is shown in Equation 6.27. Now, as the surface is traversed (as in Figure 6.18) the normal vector can be calculated as a function of *u* and *v* and the end-effector tool axis can be aligned along this direction.

$$\mathbf{N}(u, v) = \mathbf{T}_1 \times \mathbf{T}_2 \qquad \textbf{6.27}$$

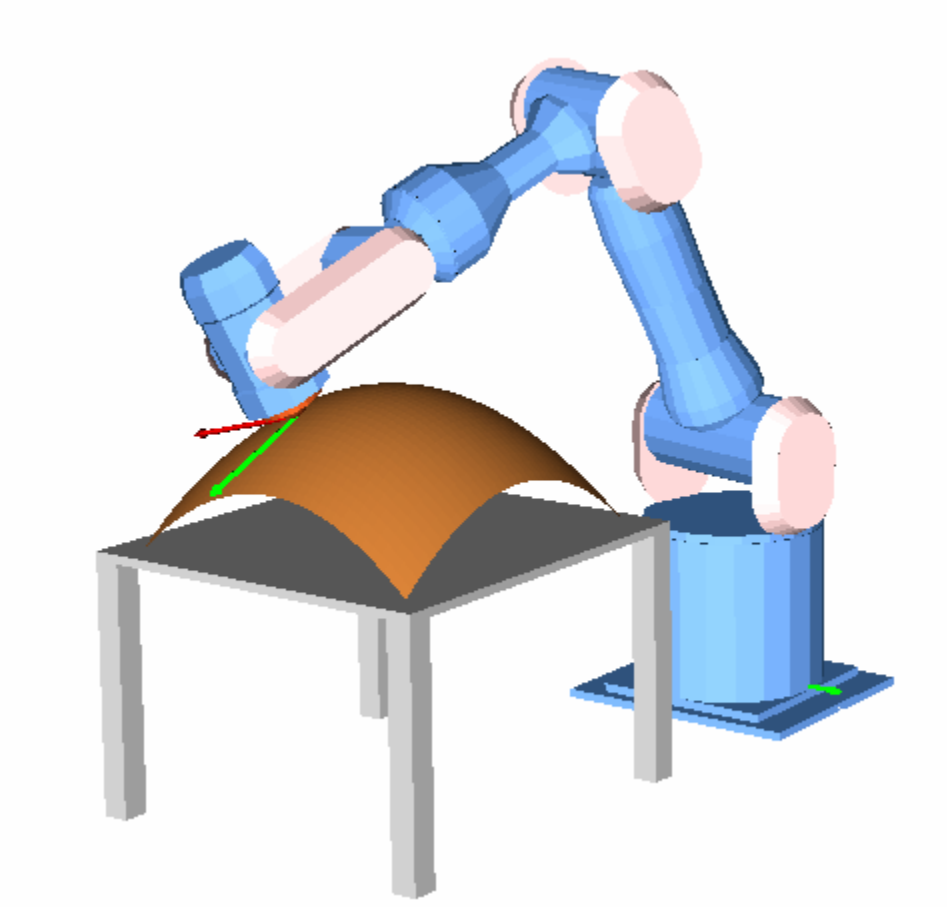


Figure 6.18. Tracing a Parabolic Surface

6.4. SOFTWARE INTEGRATION

The previous sections dealt with some of the specific implementation issues involved with the techniques developed in this research. This section will discuss the overall structure of the software integration. The first part of this implementation is to define the interface for specifying the geometric constraints. This is done by creating a *CurveParameter* object as shown in the source code in Figure 6.19. Basically, the desired frame (position and orientation) is set by a 4x4 transformation matrix, and the desired geometric constraints are supplied as numerical values. Then, any number of *CurveParameter* objects can be strung together to form an overall path. For example, two of these objects would represent a one segment curve, and three of these objects would define a two segment curve. The *orientationMode* parameter defines how the rotational motion will be defined. The available options in this implementation are shown in Table 6.3.

```
vector<CurveParameter> ctrlPoints;
CurveParameter viaPoint;
viaPoint.frame=finalHand;
viaPoint.kappa=20;
viaPoint.tau=0.0;
viaPoint.dkappa=0.0;
viaPoint.dtau=0.0;
viaPoint.orientationMode = FrameBased;
viaPoint.isCusp = false;
viaPoint.inflection = false;
ctrlPoints.push back(viaPoint);
```

Figure 6.19. Example Code for setting Constraints

Mode	Description
Fixed	End-effector orientation will remain fixed during this trajectory segment.
FrameBased	Orientation will smoothly interpolate to a frame defined by the next Frenet Frame description.
CustomOrient	Orientation will smoothly interpolate to a custom rotational frame provided by the user.
FrenetBased	Orientation will be aligned with the Frenet Frame as the curve is transversed.

Table 6.3. Orientation Interpolation Modes

Once the geometric constraints for the various frames have been defined, a manipulator motion can be commanded in a similar way to the methods described in Section 6.2.4 earlier in this chapter. An example of this is shown in Figure 6.20.

```
double moveTime=5.0;
if (!motionPlanner.PlanMoveViaGeometric(currentJoints,
                                        ctrlPoints,
                                        moveTime)){
  DisplayError(motionPlanner.GetError());
  return 0;
}

do{
  if (!motionPlanner.GetJointPosition(currentJoints,
                                      jointVel,
                                      state)){
    DisplayError(motionPlanner.GetError());
    break;
  }
  SetJoints(currentJoints);

}while(state != TrajectoryGenerator::TrajectoryComplete);
```

Figure 6.20. Example Code for Executing a Trajectory

In the above code, it can be seen that the execution of the MP has two main functions: PlanMoveViaGeometric(…) and GetJointPosition(…). In the first function, all of the computationally complex calculations for defining the curve take place. Then, the GetJointPosition(…) simply computes and returns the next joint position. The calculations are separated in this way to ensure that the GetJointPosition(…) can execute at a real-time rate. Specifically, the steps taken inside the PlanMoveViaGeometric(…) are as follows:

1. The provided geometric constraints ($(\kappa, \kappa', \tau, \tau')$ are converted into parametric constraints $\left(\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^n x}{du^n}, \frac{d^n y}{du^n}, \frac{d^n z}{du^n}\right)$ for each trajectory segment using the techniques developed in Chapter 5. There will be n-1 segments for n defined via points.

2. For each individual coordinate, the necessary coefficients are calculated to produce $\left[x(u)\ y(u)\ z(u)\right]$. This fully describes the *geometry* of the path.
3. The arc length for the entire spatial curve is calculated by discretely sampling the curve and summing the distances between points.
4. This arc length along with the desired trajectory execution time can be used to define the desired velocity profile as discussed earlier in this chapter.

The above steps define the pre-calculations done at the beginning of a trajectory generation and do not need to be placed inside of any kind of real-time loop. This is necessary as some of these calculations (such as computing the arc length) are computationally complex. Then, using the descriptions above, at each sample period (i.e. GetJointPosition(…) call), the following steps take place:

1. The current commanded velocity, $v(t)$, can be calculated from the defined velocity profile.
2. The value of the geometric parameter *u* can be calculated using Equation 6.15.
3. The positional coordinates (*x*, *y*, *z*) can then be found as they are functions of *u* ($\left[x(u)\ y(u)\ z(u)\right]$).
4. The orientation coordinates can then be defined based on the orientation interpolation scheme (Table 6.3).
5. The desired end-effector position (translational and rotational components) are sent to the Inverse Kinematics routines to convert them into a joint position. This joint position can then be sent to the manipulator controller or simulator.

In addition to the trajectory definition/execution described above, the MP also has low-level functionality for checking joint limit violations. Thus, if a joint position or velocity limit is approached, the MP will return an error and stop the trajectory execution. The full API for the MP is included in Appendix C.

6.5. SUMMARY

This chapter dealt with the details of implementing the path generation techniques developed in this work onto a physical system. This began by describing the framework for an already existing OSCAR-based Motion Planner. Then, methods for defining a speed-controlled interpolation along a parametrically defined geometric path were presented. The results of this study showed that a second-order approximation was adequate to provide good control over the speed of the trajectory. Finally, an introduction to rotational motion planning was provided. This study showed several different methods for rotational interpolation that can be currently used inside the MP as well as an introduction to a more geometric-based approach that will be an important part of future work.

These techniques were all successfully integrated into the existing OSCAR-based Motion Planner software. This MP provides a generalized, robot independent architecture for providing basic motion planning for any serial manipulator. This integration provides a useful test bed for future work in that the techniques developed in this work can be directly applied and tested on a variety of robotic systems. However, there remain a number of improvements that can be made both to the low-level curve generation techniques as well as the high-level software implementation. In the next chapter, the results of this work will be summarized and then suggestions for future work will be presented.

7. CHAPTER SEVEN

Summary and Future Work

In this report, a new method for describing spatial paths for manipulator motions was developed using the geometric properties of curvature and torsion. This began by looking at the basic mathematics of algebraic curves. Then, a brief review of some current interactive curve generation techniques from other disciplines was presented. Next, a thorough study of the affects of curvature on the local geometry of a curve was conducted by creating local surfaces with families of curves. Then, a curve generation technique was developed to utilize these properties. This works by converting the geometric constraints (curvature, torsion, and their derivatives) into parametric constraints up to the fourth order $\left(\frac{dx}{du},\frac{dy}{du},\frac{dz}{du},\cdots,\frac{d^4x}{du^4},\frac{d^4y}{du^4},\frac{d^4z}{du^4}\right)$ that can be more easily blended together because of their Cartesian nature. Finally, these techniques were integrated into an existing motion planning software architecture built using OSCAR. This chapter will provide a summary of the work described above as well as a demonstration of its use. Finally, suggestions for future work and applications of this research will be presented..

7.1. SUMMARY

7.1.1. Algebraic Curves

As mentioned in the previous section, this research began with a study of the basic mathematics of algebraic curves. This was done by examining four different possible representations for curves: implicit, standard parametric, arc-length parametric, and curvature/torsion profiles. Table 7.1 shows a summary of these four different

representations. Then, a list of the advantages and disadvantages of each is shown in Table 7.2.

	Planar	Spatial
Implicit	$f(x,y)=0$	$f(x,y,z)=0 \cap g(x,y,z)=0$
Standard Parametric	$\left.\begin{aligned} x &= f(u) \\ y &= f(u) \end{aligned}\right\} u \in [a,b]$	$\left.\begin{aligned} x &= f(u) \\ y &= f(u) \\ z &= f(u) \end{aligned}\right\} u \in [a,b]$
Arc Length Parametric	$\left.\begin{aligned} x &= f(s) \\ y &= f(s) \end{aligned}\right\} s \in [a,b]$	$\left.\begin{aligned} x &= f(s) \\ y &= f(s) \\ z &= f(s) \end{aligned}\right\} s \in [a,b]$
Curvature/Torsion Profile	$\kappa = f(s)$ $\tau = 0$	$\kappa = f(s)$ $\tau = f(s)$

Table 7.1. Curve Representations

	Advantages	Disadvantages
Implicit	• Good mathematical understanding of singularities (double points, cusps, etc) • Historical literature and research	• Becomes increasingly complex as curve degree gets higher • Difficult to represent in spatial form • Difficult to describe an actual motion along its arc length
Standard Parametric	• Provides a one-to-one mapping from $R \to R^3$ • Easy to define in a finite interval as for piecewise segments • Easy to define in spatial form	• Lack of physical meaning in term of the independent parameter • Some loss of mathematical understanding compared to implicit forms

Arc Length Parametric	• Provides good physical meaning to independent parameter • Easy to define physical motion along curve • Calculation of some curve properties becomes easier	• Difficult to find closed-form solutions for most curves • Numerical techniques needed
Curvature/ Torsion Profile	• Defines curve based on higher-order properties • Geometric shape is independent of position/orientation	• Difficult to define global motions • Best used for defining local geometry

Table 7.2. Comparision of Curve Representations

The table presented above shows that every form of curve representation has certain advantages depending on the application. Thus, an effort was put forth in this work to retain as many representations as possible. For example, Figure 7.1 shows two simple planar curves represented both implicitly and parametrically.

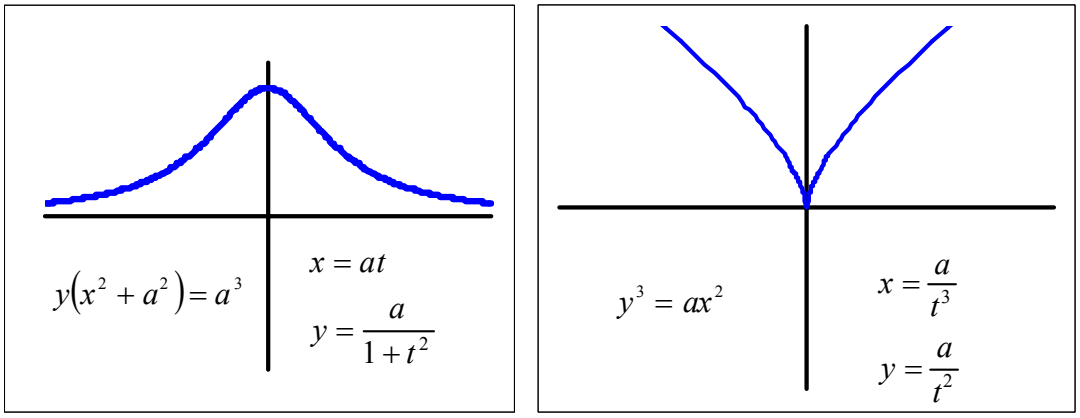


Figure 7.1. Examples of Algebraic Curves

However, the representation most commonly used, in this work as well as the literature, is the standard parametric form. Thus, it is worth further summarizing this representation (see Section 2.2 for full discussion). In this form, the curve is defined as function of some independent parameter represented[12] here by the symbol u. A curve is then defined on some finite interval [a,b] of u as $\mathbf{p}(u) = \left[x(u)\ y(u)\ z(u) \right]$, $u \in [a,b]$. Thus, each scalar value of u maps to a spatial point location $[x,y,z]$. The curvature of a parametic curve can then be defined as in Equation 7.1, where $x' = dx/du$. This equation shows that curvature is a function of the first and second order derivatives of the parametric description.

$$\kappa(u) = \frac{\sqrt{(y'z'' - y''z')^2 + (z'x'' - x'z'')^2 + (x'y'' - y'x'')^2}}{(x'^2 + y'^2 + z'^2)^{3/2}} \qquad \textbf{7.1}$$

The next important property of spatial curves is torsion. The definition of torsion is shown in Equation 7.2. This equation shows that torsion is a function of the first, second, and third order derivatives.

$$\tau(u) = \frac{(y'z''x''' - y''z'x''') + (z'x''y''' - x'z''y''') + (x'y''z''' - y'x''z''')}{(y'z'' - y''z')^2 + (z'x'' - x'z'')^2 + (x'y'' - y'x'')^2} \qquad \textbf{7.2}$$

Another important property of spatial parametric curves is the Frenet Frame. This is a three dimensional orthogonal frame defined by the local geometry of the curve. It consists of three vectors: the unit tangent, unit normal, and unit bi-normal. The unit tangent basically represents the "heading" of the curve and is calculated as shown in Equation 7.3.

$$\hat{\mathbf{T}}(u) = \frac{[x' \ \ y' \ \ z']}{\sqrt{x'^2 + y'^2 + z'^2}} \qquad \textbf{7.3}$$

[12] The independent parameter is often represented as t in mathematics literature. However, in this work, the variable t is reserved to represent time.

Then, the unit normal can be calculated by taking the derivative of the unit-tangent as shown in Equation 7.4. Finally, the bi-normal is calculated by taking the cross product of the tangent and normal (Equation 7.5).

$$\hat{\mathbf{N}}(u) = \frac{\frac{d\hat{\mathbf{T}}}{du}}{\left\| \frac{d\hat{\mathbf{T}}}{du} \right\|} \tag{7.4}$$

$$\hat{\mathbf{B}}(u) = \hat{\mathbf{T}}(u) \times \hat{\mathbf{N}}(u) \tag{7.5}$$

A useful mathematic relationship can be developed from the Frenet Frame known as the Frenet-Serret formulas (Equation 7.6). This shows that the motion of the Frenet Frame along the curve can be defined locally in terms of the curve curvature and torsion. This relationship can thus be integrated to define curves in terms of curvature and torsion. This result is used later in this work to help provide a more physical understanding of these properties.

$$\begin{bmatrix} \frac{d\hat{\mathbf{T}}}{ds} \\ \frac{d\hat{\mathbf{N}}}{ds} \\ \frac{d\hat{\mathbf{B}}}{ds} \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}} \\ \hat{\mathbf{N}} \\ \hat{\mathbf{B}} \end{bmatrix} \tag{7.6}$$

In this section, the basic properties of parametric curves were presented. These properties and their physical meanings are more thoroughly investigated in the later portions of this research. However, the descriptions in this section provide a useful starting point for understanding the rest of this work.

7.1.2. Interactive Curve Generation Techniques

After a basic understanding of algebraic curves has been reached, a look into some of the techniques from other disciplines (such as Computer-Aided Design and

Computer Graphics) was provided (see Chapter 3). In this summary, a few of these techniques will be described and then discussed. One of the simpler representations discussed in this review was B-Splines. B-Splines are described by a set of control points and the B-Spline basis functions. The basic equation for a B-Spline is shown in Equation 7.7 where the basis functions $N_{i,k}(u)$ are calculated recursively as shown in Equation 7.8.

$$\mathbf{p}(u) = \sum_{i=1}^{n} \mathbf{b}_i N_{i,k}(u) \tag{7.7}$$

$$N_{i,k}(u) = \frac{N_{i,k-1}(u)(u-u_i)}{(u_{i+k-1}-u_i)} + \frac{N_{i+1,k-1}(u)(u_{i+k}-u)}{(u_{i+k}-u_{i+1})} \tag{7.8}$$

An example of a B-Spline is shown in Figure 7.2. This shows that the shape of the B-Spline loosely follows the shape of its control polygon. Thus, interactively, B-Splines are often generated by moving around or adding/deleting control points to generate the desired curve geometry.

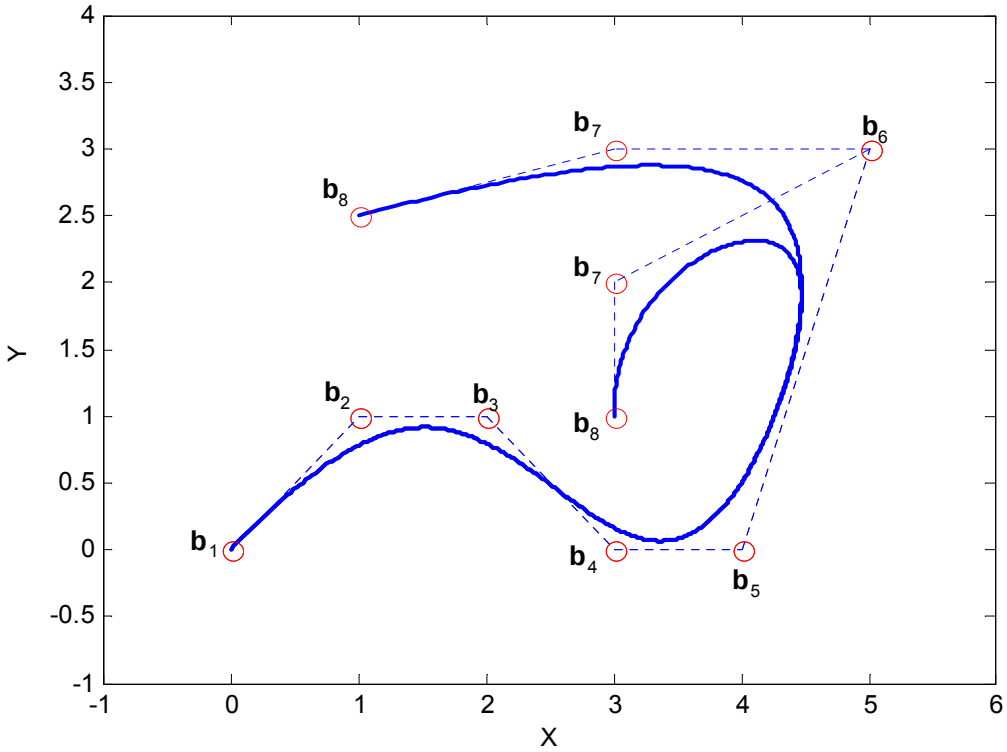


Figure 7.2. Example B-Spline Curves

In addition to providing a good way to intuitively design curves, B-Splines have several additional advantageous properties: convex hull, variation diminishing, and local control. These properties basically ensure that the B-Spline curve is "well-behaved" although its higher-order properties may be less controllable. A more detailed description of this technique and its properties was presented in Section 3.2.2.

Another method of curve design studied in this work is Algebraic Splines (A-Splines) [5][6]. A-Splines are defined implicitly using barycentric coordinates as shown in Equation 7.7 where $B_{ijk}^{n}\left(\alpha_1,\alpha_2,\alpha_3\right)=\frac{n!}{i!\,j!\,k!}\alpha_1^i\alpha_2^j\alpha_3^k$. By using barycentric coordinates, the A-Splines are able to define a higher level of geometric continuity (managed higher-order properties) with more degrees of freedom. An example of a family of G^1 continuous A-Splines is shown in Figure 7.3.

$$F\left(\alpha_1,\alpha_2,\alpha_3\right)=\sum_{i+j+k=n} b_{ijk}B_{ijk}^{n}\left(\alpha_1,\alpha_2,\alpha_3\right)=0 \qquad \textbf{7.9}$$

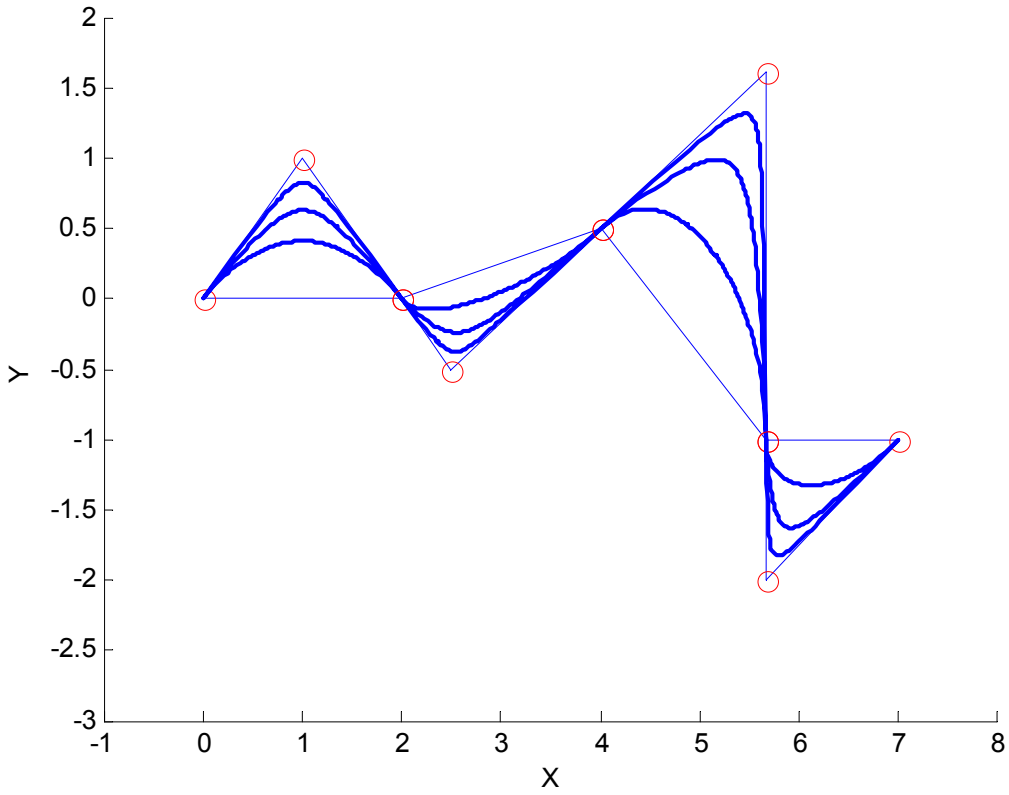


Figure 7.3. A-Spline Curve Example

As mention in Section 3.4, A-Splines have several advantages. First, they offer a higher degree of freedom for creating/designing curves. Second, they offer the ability to capture both parametric and implicit forms of curves. However, they require numerical techniques to trace (i.e. describe a motion along) for higher orders, and it is difficult to describe spatial curves/motions.

The techniques described in this review can provide powerful and intuitive methods for designing visually pleasing curves. However, the main focus of this research is on defining local geometric constraints with a well-understood physical meaning. Thus, the next step in this research is to further examine the relationships between curvature/torsion and the local geometry of curves. However, in the future, it may be possible to adapt the methods described in this section for the purposes of blending between these local constraints. The benefit of this merging of techniques would be to generate curves with well-defined properties (e.g. Convex Hull or Variation Diminishing). This will be briefly explored in the section on future work.

7.1.3. Geometric Shapes and Properties

In order to study the physical meaning of curvature/torsion and their derivatives, specific geometric shapes were analyzed. This study began with simple planar shapes such as lines, circles, and parabolas. This involved developing both the implicit and parametric forms of these curves and identifying parameters that could be used to generate families of curves. Then, closed-form solutions for curvature[13] were found in terms of these parameters. For example, consider the simple parabola described by Equation 7.10. The parameter *a* can then be varied to provide a family of curves as shown Figure 7.4.

[13] By definition, torsion is always zero for a planar curve. Thus, for planar curves, only curvature was studied.

$$y - ax^2 = 0 \tag{7.10}$$

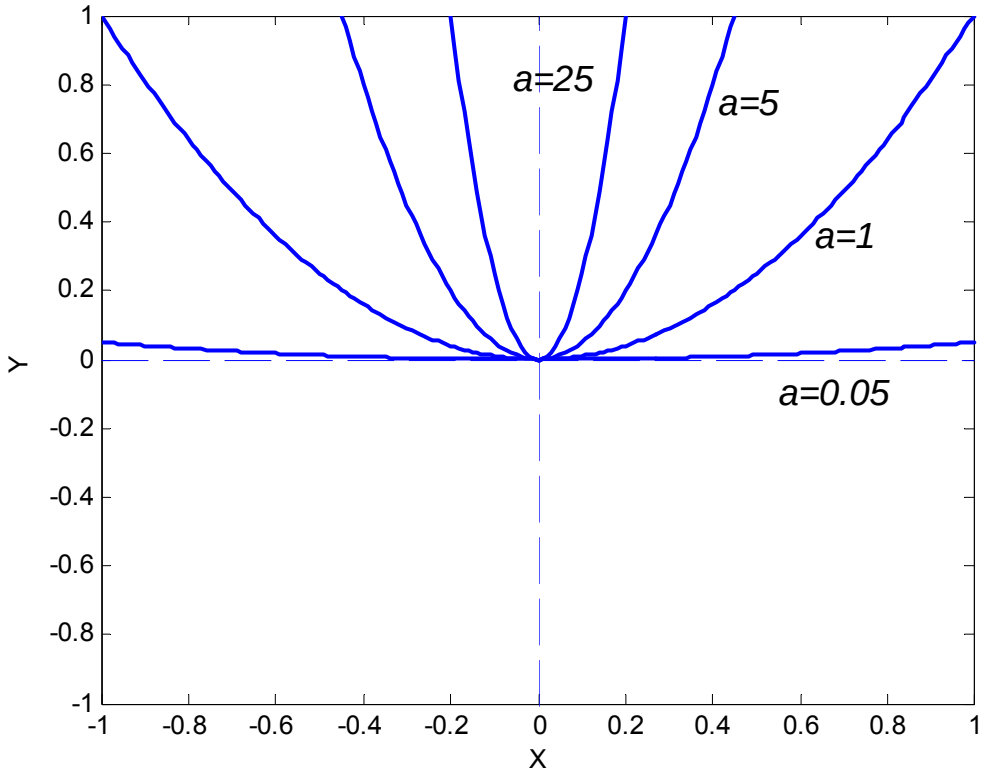


Figure 7.4. Family of Parabolas

Now, this implicit equation can be substituted into the equation for curvature to calculate a closed-form solution for curvature in terms of the parameter *a*. This is shown in Equation 7.11. From this equation, it is easy to see that the maximum curvature occurs at the origin and has a magnitude of 2*a*. This is intuitive from the above plot as the maximum bending can be seen to occur at the origin. Table 7.3 summarizes the results of Figure 7.4. While this example is very simple, it provides some insight into the relationship between curvature and simple geometric shapes.

$$\kappa(x,y) = \frac{-2a}{\left(1+4a^2x^2\right)^{3/2}} \tag{7.11}$$

a	Implicit Equation	Parametric Equation	$\lvert\kappa_{max}\rvert$ (κ at origin)
0.05	$y - 0.05x^2$	$x(u) = u$ $y(u) = 0.05u^2$	0.1
1	$y - x^2$	$x(u) = u$ $y(u) = u^2$	2
5	$y - 5x^2$	$x(u) = u$ $y(u) = 5u^2$	10
25	$y - 25x^2$	$x(u) = u$ $y(u) = 25u^2$	50

Table 7.3. Summary of Family of Parabolas

A similar analysis was conducted for a variety of planar shapes. In each case, the implicit and parametric forms of the equation were presented. Parameters were identified that could be used to generate families of these curves. Then, curvature was solved for in terms of these parameters. This provides insight into the relationship between these simple shapes and the intrinsic property of curvature. A summary of these results is shown in Table 7.4.

Shape	Implicit Equation	Parametric Equation	$\kappa(x,y)$
Line	$ax+by+c=0$	$x(u)=a_1u+b_1$ $y(u)=a_2u+b_2$	0.0
Parabola	$y-ax^2=0$	$x(u)=u$ $y(u)=au^2$	$\dfrac{-2a}{\left(1+4a^2x^2\right)^{3/2}}$
Circle	$x^2+y^2-r^2=0$	$x(u)=\dfrac{r\left(1-u^2\right)}{1+u^2}$ $y(u)=\dfrac{2ur}{1+u^2}$	$1/r$
Ellipse	$\dfrac{x^2}{a^2}+\dfrac{y^2}{b^2}-1=0$	$x(u)=\dfrac{-\left(\dfrac{2}{b}\right)u}{\dfrac{1}{a^2}+\left(\dfrac{1}{b^2}\right)u^2}$ $y(u)=\dfrac{-\left(\dfrac{1}{b}\right)u^2+\left(\dfrac{b}{a^2}\right)}{\dfrac{1}{a^2}+\left(\dfrac{1}{b^2}\right)u^2}$	$\kappa(x,y)=\dfrac{8AB}{\left(4A^2x^2+4B^2y^2\right)^{3/2}}$
Cusp	$ay^2-x^3=0$	$x(u)=au^2$ $y(u)=au^3$	$\kappa(x,y)=\dfrac{18ax^4-24a^2xy^2}{\left(9x^4+4a^2y^2\right)^{3/2}}$

Table 7.4. Summary of Properties of Planar Shapes

The next step of this analysis involved expanding this study into the spatial domain. One of the most basic spatial shapes is the helix. A helix is defined by a constant curvature and torsion along its path. A good way to visualize these curves is to show them running along the surface of a cylinder. This is shown in Figure 7.5 for three sets of curvature/torsion values.

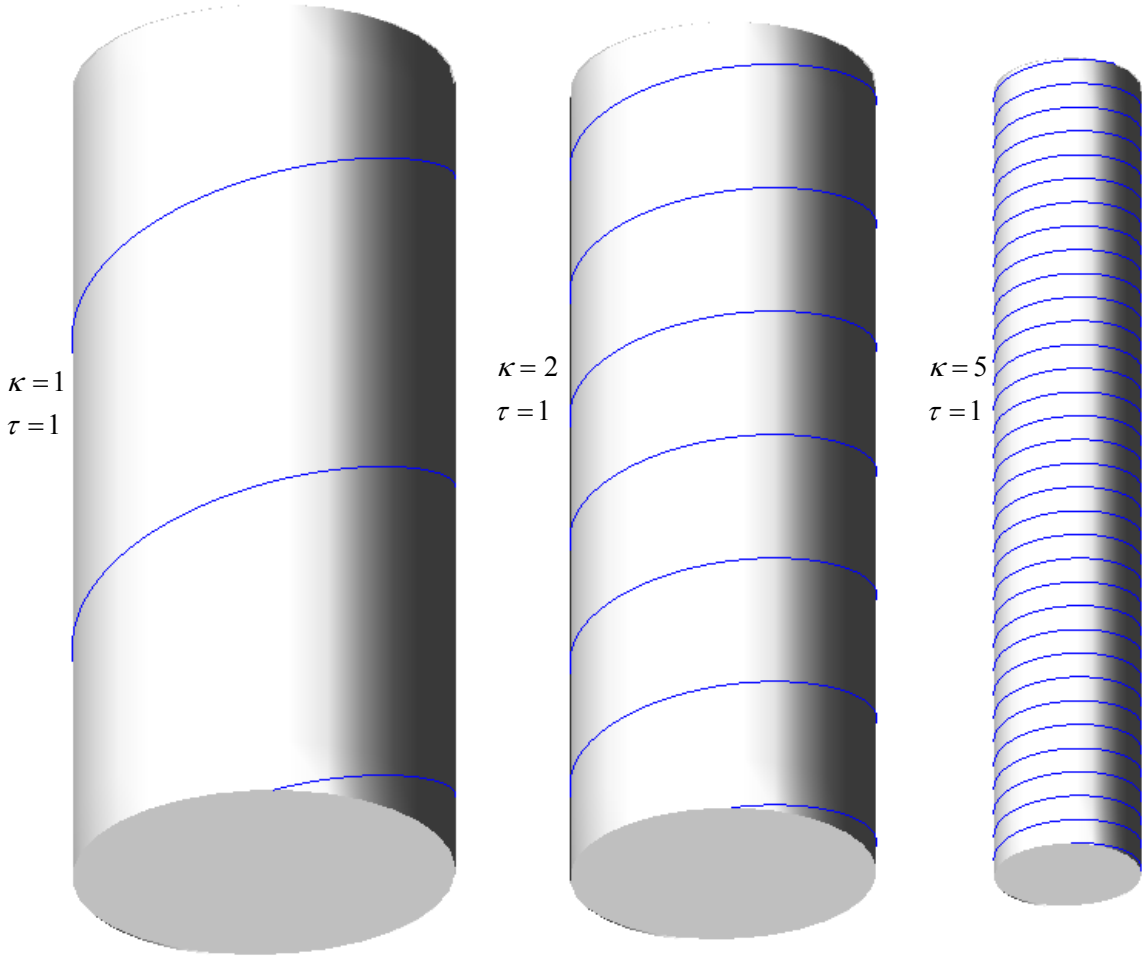


Figure 7.5. Constant Curvature/Torsion Curves

The above plot shows that as the curvature increases the radius of the cylinder decreases and the distance travelled along each wrap around (i.e. the "pitch") becomes smaller. In fact, the relationship between curvature/torsion and radius/pitch can be directly solved and is shown in Equations 7.12 [26]. These relationships can be easily inverted to solve for r and l in terms of curvature and torsion as shown in Equation 7.13 (see Section 4.4.1). Table 7.5 provides a summary of these results.

$$\kappa = \frac{r}{r^2 + l^2}, \tau = \frac{l}{r^2 + l^2} \quad \textbf{7.12}$$

$$r = \frac{\kappa}{\kappa^2 + \tau^2}, l = \frac{\tau}{\kappa^2 + \tau^2} \quad \textbf{7.13}$$

κ	τ	$\kappa^2+\tau^2$	**r**	**l**	**Parametric Equation**
1	1	2	0.5	0.5	$x(u)=0.5\cos(u)$ $y(u)=0.5\sin(u)$ $z(u)=0.5u$
2	1	5	0.4	0.2	$x(u)=0.4\cos(u)$ $y(u)=0.4\sin(u)$ $z(u)=0.2u$
5	1	26	0.1923	0.0385	$x(u)=0.1923\cos(u)$ $y(u)=0.1923\sin(u)$ $z(u)=0.0385u$

Table 7.5. Properties of Helical Curves

However, not all spatial shapes can be easily described in terms of curvature and torsion. Thus, a main part of the study of spatial curves in this research was achieved by directly generating curves based on their local curvature and torsion values. This allows the values of curvature and torsion to be modified to generate local families of curves based on these parameters. These curves are based on integrating the Frenet-Serret formulas to generate the local shape of a curve around a provided frame. To perform this, a frame is first placed at the origin with the *x, y, z* axes lined up with the tangent, normal, and bi-normal directions respectively. Then, given a provided curvature/torsion profile, a curve can be generated in the "forward" and "reverse" direction around the local frame by using the formulation shown in Equation 7.14 (see Section 4.4). It should be noted that the position/alignment of the frame is arbitrary as a given curvature/torsion will always produce the exact same motion relative to the frame.

$$\begin{aligned}\mathbf{T}_{i+1} &= \mathbf{T}_i + \kappa \mathbf{N}_i \Delta s \\ \mathbf{B}_{i+1} &= \mathbf{B}_i - \tau \mathbf{N}_i \Delta s \\ \mathbf{N}_{i+1} &= \mathbf{B}_{i+1} \times \mathbf{T}_{i+1} \\ \mathbf{P}_{i+1} &= \mathbf{P}_i + \mathbf{T}_{i+1} \Delta s\end{aligned} \qquad \textbf{7.14}$$

A simple example of this is shown in Figure 7.6 for varying curvature and zero torsion. This shows that the curve will remain in the plane defined by the tangent/normal vectors (the *xy* plane in this example), and the higher curvature values will result in a sharper bend around the normal vector. It remains symmetric in the *y*-*z* plane.

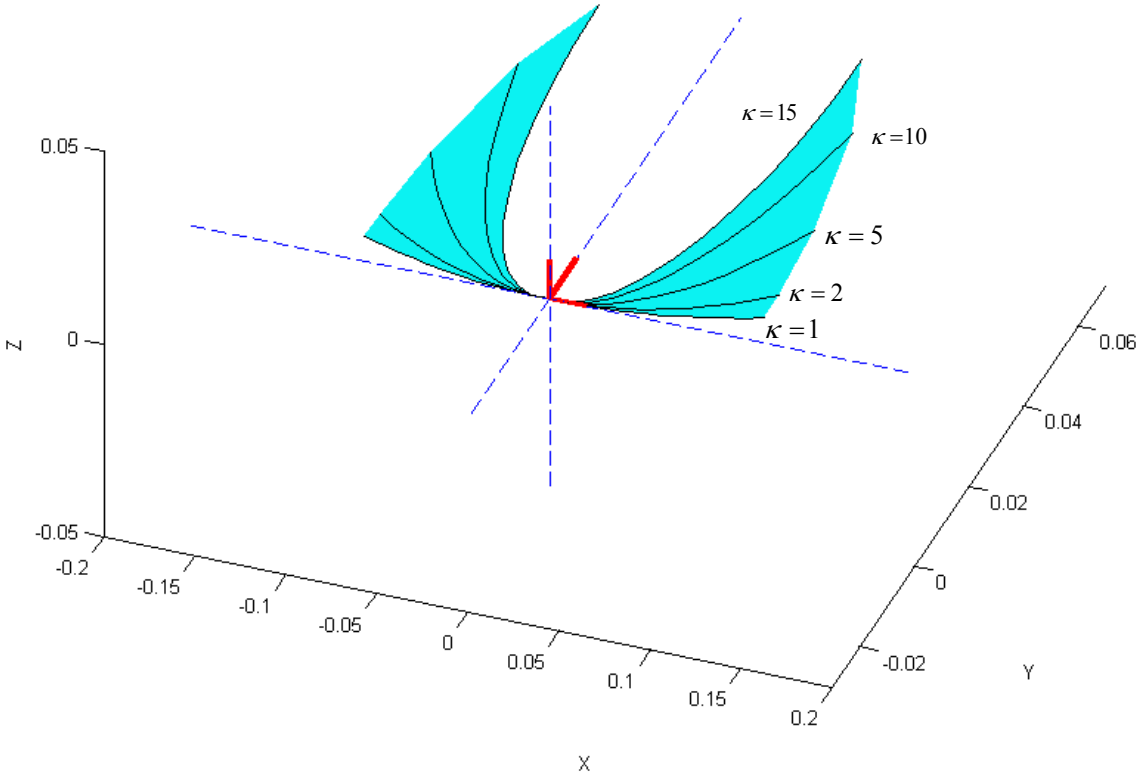


Figure 7.6. Local Effect of Varying Curvature

A similar set of figures can be developed to describe the local effects of torsion. This is shown in Figure 7.7 and Figure 7.8. Unlike curvature, which is always defined to be positive, torsion can be either positive or negative. Locally, the sign of this value

dictates whether the curve leaves the plane in the direction of the bi-normal vector or its inverse direction. For example, in Figure 7.7, the curve approaches and leaves the local frame while moving in the positive z directions. On the other hand, in Figure 7.8, the curve changes directions with respect to the z axis at the frame. Another thing to note about the affects of torsion is that the scale of the z axis in these plots is much smaller than the x and y axes. This is because, numerically, torsion has a smaller effect on the shape of the curve than an equivalent value of curvature.

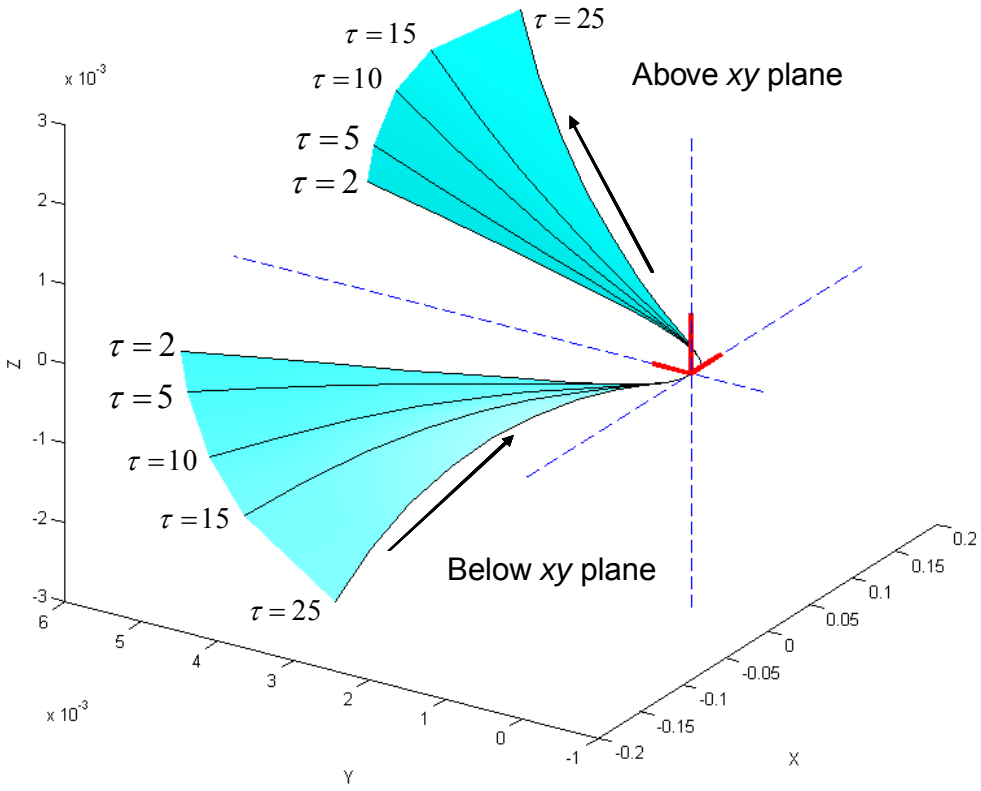


Figure 7.7. Varying Positive Torsions

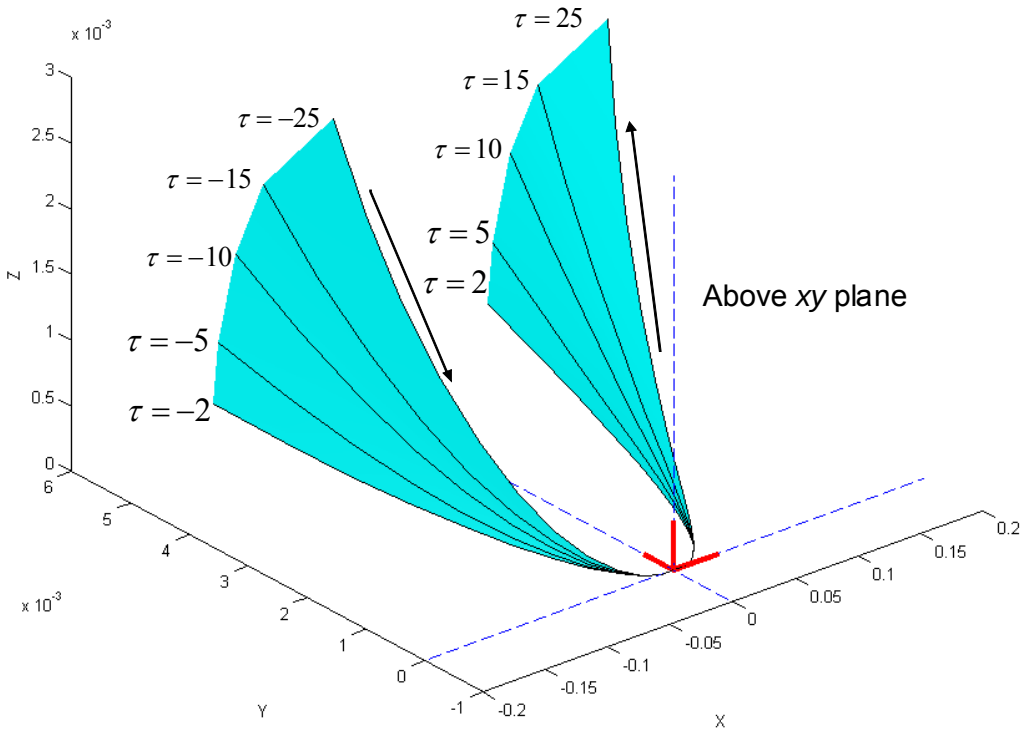


Figure 7.8. Varying Negative Torsions

Once a basic understanding of these properties is reached, this research showed how they can be used to generate spatial shapes such as cusps (Figure 7.9) and saddle points (Figure 7.10). This varies from the approach taken to planar shapes in that instead of describing the shapes based on implicit/parametric equations and then calculating curvature, the curves are actually described in terms of curvature and torsion. Then, the next step of this research is to convert these properties back into a parametric form that can be used to create spatial curves bounded by these local constraint parameters (end-of-motion specifications). This is the focus of the next section.

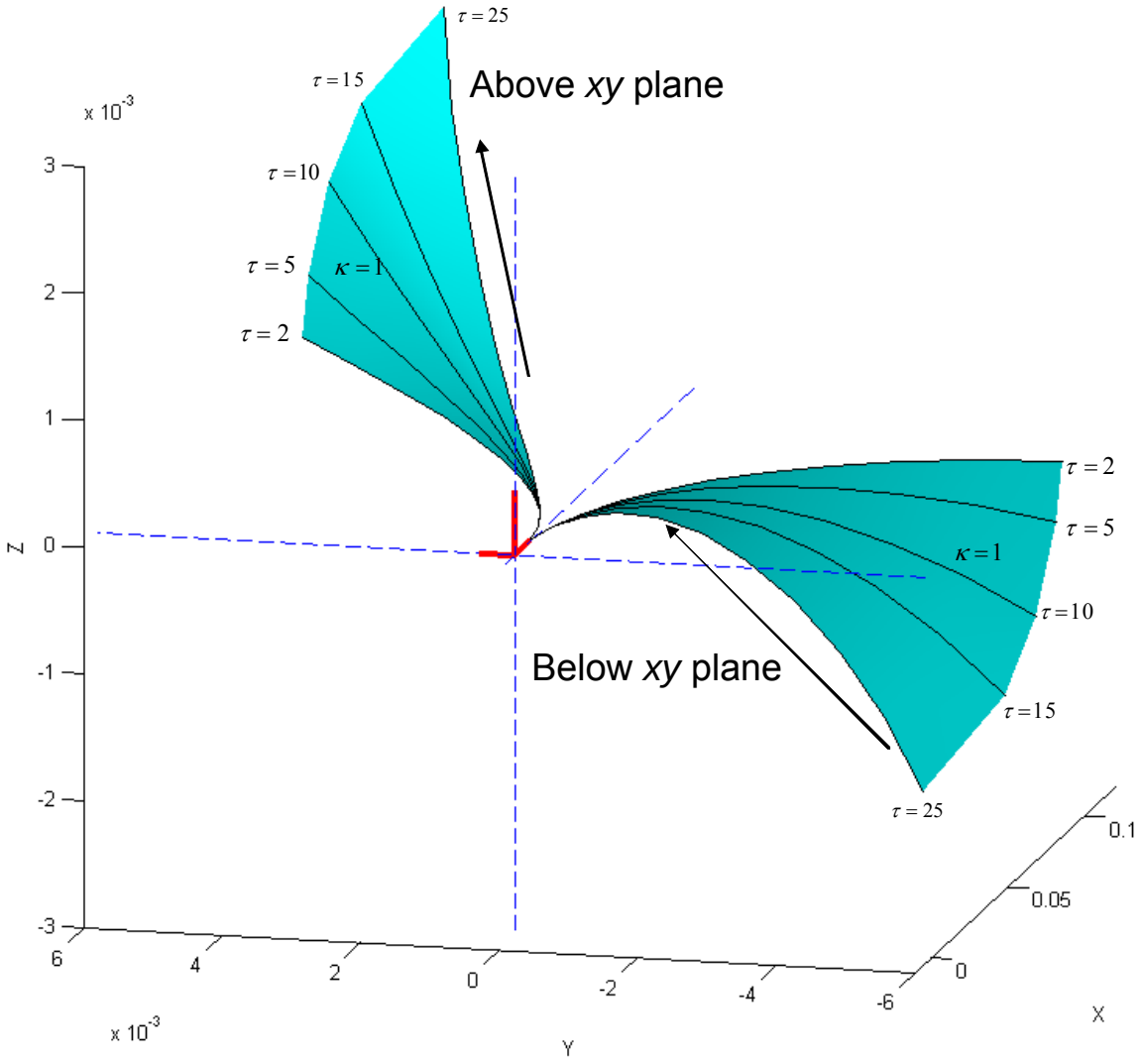


Figure 7.9. Local Cusp

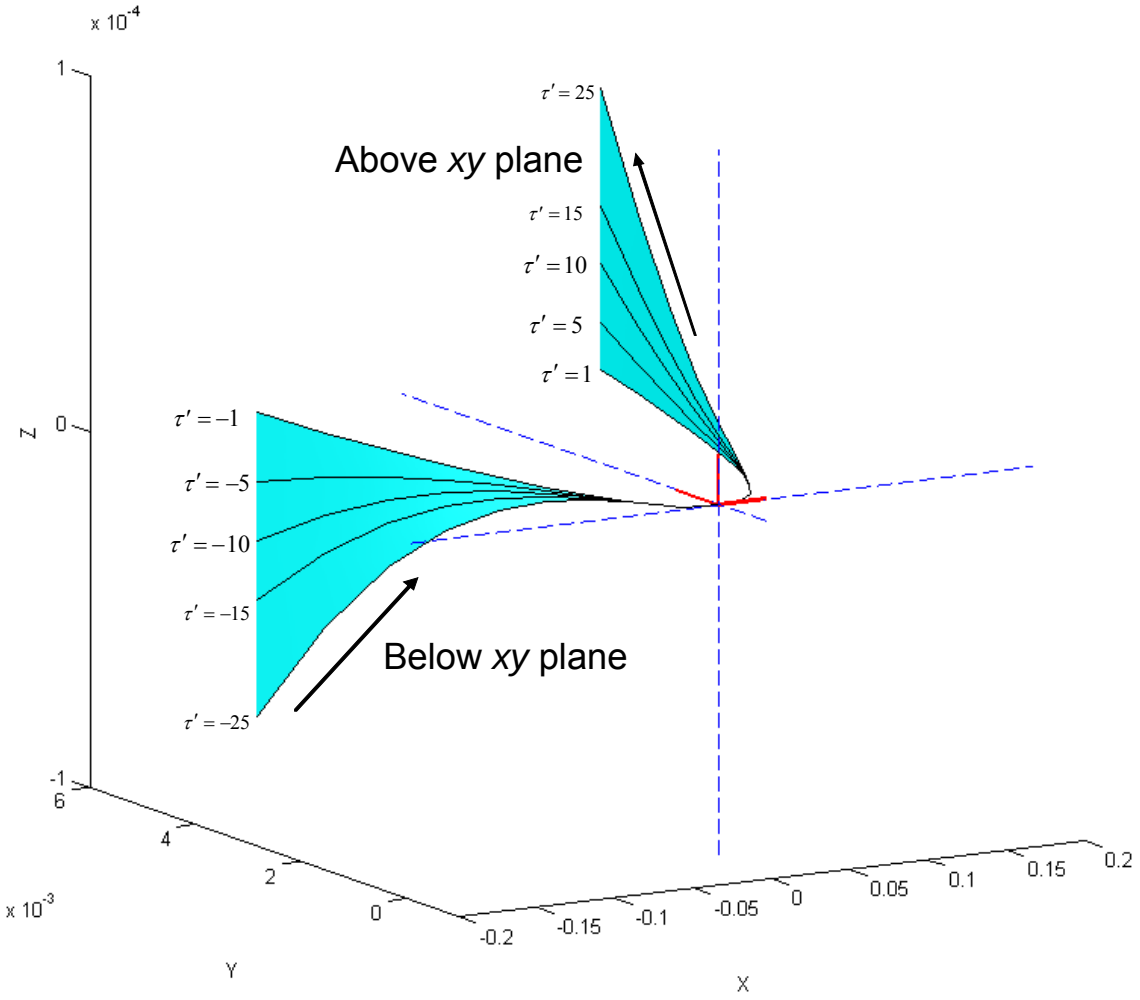


Figure 7.10. Local Saddle Point

7.1.4. Path Generation with Geometric Constraints

As mentioned in the previous section, the next step of this work focuses on converting the geometric constraints $(\kappa, \kappa', \tau, \tau')$ into parametric constraints $\left(\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^4x}{du^4}, \frac{d^4y}{du^4}, \frac{d^4z}{du^4}\right)$ that can be used to formally generate spatial curves. This section will present the main results/process of doing this. The full derivations of these results are presented in Chapter 5. The main steps behind this process are:

1. First, a set of spatial coordinate frames (positions and orientations) is defined. These would most likely come from a simulation or CAD environment and would not have to be defined by an operator.
2. The desired geometric properties at each frame are defined (κ, κ', τ, and τ').
3. The first order parametric constraint ($\frac{d\mathbf{p}}{du}$) is defined using the unit tangent vector as $\frac{d\mathbf{p}}{du} = \left\|\frac{d\mathbf{p}}{du}\right\| \hat{\mathbf{T}}(\mathrm{u})$ where $\left\|\frac{d\mathbf{p}}{du}\right\|$ is a controllable parameter
4. The second order parametric constraint ($\frac{d^2\mathbf{p}}{du^2}$) is defined using the unit normal vector and desired curvature as $\frac{d^2\mathbf{p}}{du^2} = \left\|\frac{d\mathbf{p}}{du}\right\|^2 \kappa\hat{\mathbf{N}}$
5. The third order parametric constraint ($\frac{d^3\mathbf{p}}{du^3}$) is defined using the desired torsion and derivative of curvature as $\begin{bmatrix} \tau \\ \frac{d\kappa}{du}\kappa\left\|\frac{d\mathbf{p}}{du}\right\|^6 \end{bmatrix} = \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \end{bmatrix} \begin{bmatrix} x''' \\ y''' \\ z''' \end{bmatrix}$
6. The fourth order parametric constraint ($\frac{d^4\mathbf{p}}{du^4}$) is defined using the derivative of torsion as $\frac{d\tau}{du}\left\|\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right\|^2 + 2\tau\left(\frac{d\mathbf{p}}{du}\times\frac{d^2\mathbf{p}}{du^2}\right)\cdot\left(\frac{d\mathbf{p}}{du}\times\frac{d^3\mathbf{p}}{du^3}\right) = a_0x^{(4)} + b_0y^{(4)} + c_0z^{(4)}$
7. Once the parametric constraints ($\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^4x}{du^4}, \frac{d^4y}{du^4}, \frac{d^4z}{du^4}$) have been defined, the *x*, *y*, and *z* trajectories are calculated independently using some trajectory planning technique (polynomial, trapezoidal, etc)
8. The individual trajectories are combined to produce a parametric description of the desired spatial curve $\mathbf{p}(u) = \begin{bmatrix} x(u) & y(u) & z(u) \end{bmatrix}$

For example, suppose the geometric constraints shown in Table 7.6 were provided. Using the above process, these can be converted into the parametric constraints shown in Table 7.7. Then, the overall spatial path can be developed as in Figure 7.11. It

should be noted that here we are just verifying the mathematic formulations, and a more detailed examination of the physical meaning of this process will be presented in Section 7.2.

	x	y	z	$\hat{\mathbf{T}}$	$\hat{\mathbf{N}}$	$\hat{\mathbf{B}}$	κ	τ
$\mathbf{p}_1$	0.0	0.0	0.0	[1,0,0]	[0,1,0]	[0,0,1]	0.0	0.0
$\mathbf{p}_2$	2.0	1.0	1.0	[0,0.707,-0.707]	[-1,0,0]	[0,0.707,0.707]	1.0	10.0
$\mathbf{p}_3$	1.0	3.0	-1.0	[0,1,0]	[-1,0,0]	[0,0,1]	0.0	0.0

Table 7.6. Example Geometric Constraints

i	x_i	y_i	z_i	$\frac{dx_i}{du}$	$\frac{dy_i}{du}$	$\frac{dz_i}{du}$	$\frac{d^2x_i}{du^2}$	$\frac{d^2y_i}{du^2}$	$\frac{d^2z_i}{du^2}$	$\frac{d^3x_i}{du^3}$	$\frac{d^3y_i}{du^3}$	$\frac{d^3z_i}{du^3}$
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0	1.0	1.0	0.0	0.707	-.707	-1.0	0.0	0.0	0.0	7.0679	7.0679
3	1.0	3.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 7.7. Calculated Parametric Constraints

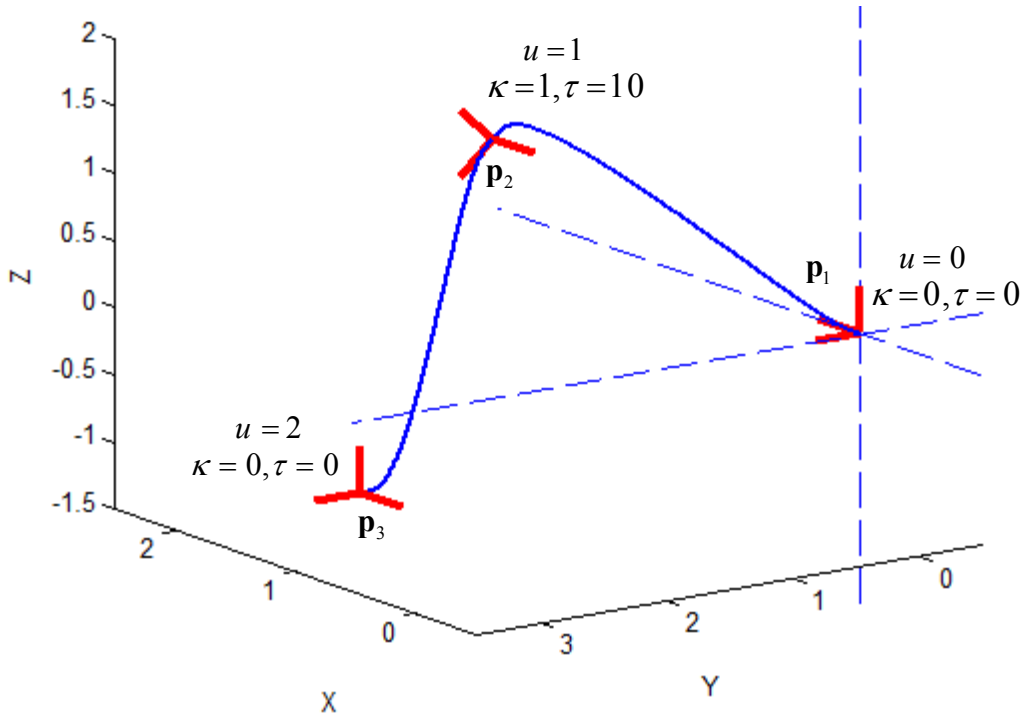


Figure 7.11. Generated Spatial Curve

7.1.5. Motion Planner Implementation

The last few sections have described the mathematical process for generating a spatial curve based on geometric properties. Once this framework was developed, the results were integrated into an OSCAR-based Motion Planner software package. A schematic overview of this is shown in Figure 7.12.

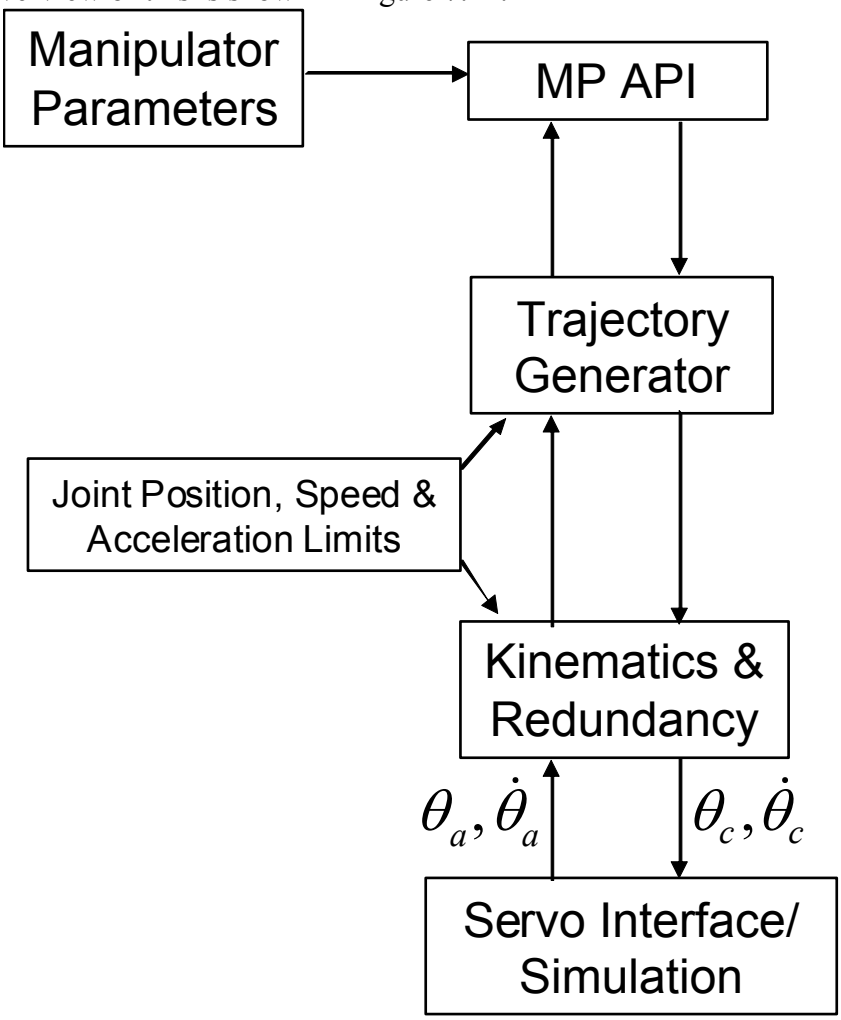


Figure 7.12. Motion Planner Schematic

One of the main issues in implementing the results of this research into an actual manipulator Motion Planner was determining how best to define $u = f(t)$. This is necessary in order to accurately control the spatial velocity along a parametrically defined path. In Chapter 6, a method that was structured in terms of an interpolater previously

designed for CNC machines [63] was described. This method uses both the geometric properties of the curve as well as the physical motion properties (velocity and acceleration) to determine a value of the geometric parameter u at every sample period. This formulation is shown in Equation 7.15. This equation shows how u can be stepped forward at each sampling period using the first and second order parametric values along with the desired velocity and acceleration. A more detailed description of this formulation can be found in Section 6.3.2.4. The resulting output velocity profile is shown in Figure 7.13.

$$u_{i+1} = u_i + \Delta t\left(\frac{v(t)}{\left\|\frac{d\mathbf{p}_i}{du}\right\|}\right) + \frac{\Delta t^2}{2}\left(\frac{a(t)}{\left\|\frac{d\mathbf{p}}{du}\right\|} - \frac{v(t)^2\left(\frac{d\mathbf{p}}{du}\cdot\frac{d^2\mathbf{p}}{du^2}\right)}{\left\|\frac{d\mathbf{p}}{du}\right\|^4}\right) \quad \textbf{7.15}$$

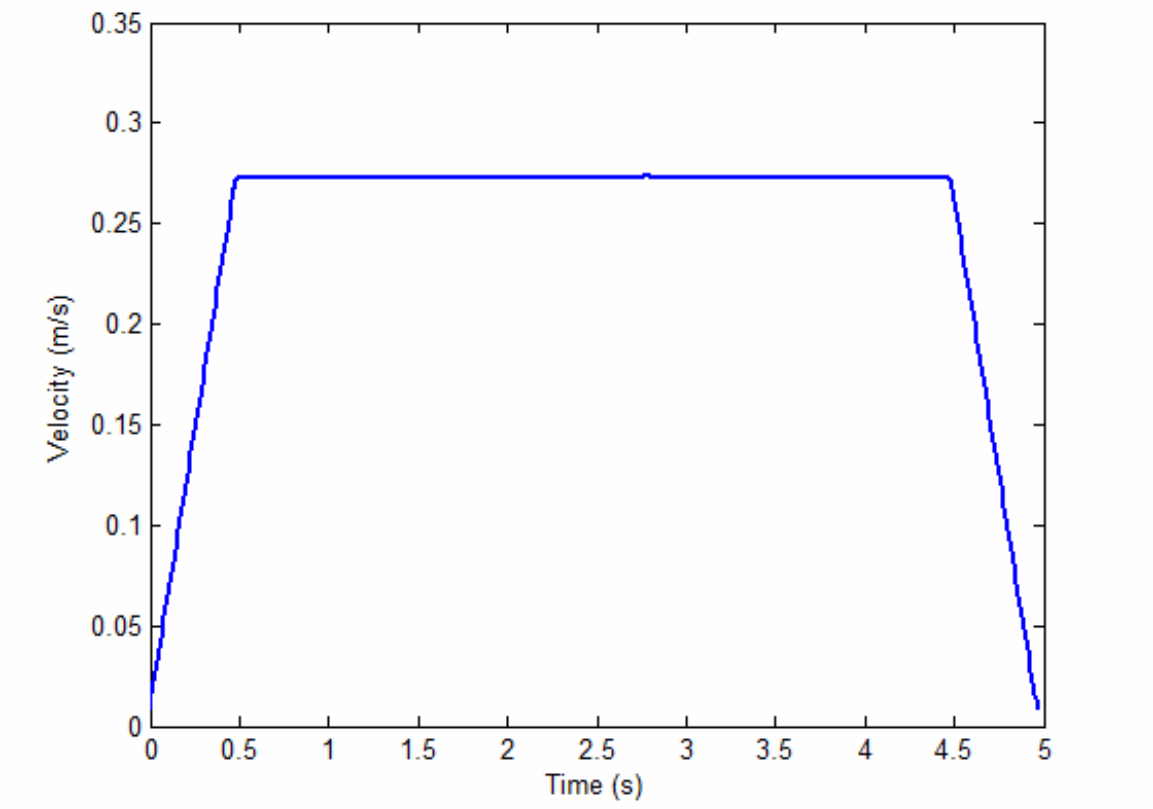


Figure 7.13. Resulting Velocity Profile

Once a good method for describing the motion along a curve was defined, this method, along with the low-level curve generation, was integrated into the existing Motion Planner architecture. This allows a user to quickly and easily use the methods generated in this work and apply them to a variety of mechanical systems. A small example of sample code used to generate a motion using this software is shown in Figure 7.14. This code has two main functions: PlanMoveViaGeometric() and GetJointPosition(). In PlanMoveViaGeometric(), the parameters and coefficients necessary to build a parametric description of the curve based on the geometric constraints are calculated. Then, GetJointPosition() can be called to find the joint positions that will step the manipulator along the calculated path.

```
double moveTime=5.0;
if (!motionPlanner.PlanMoveViaGeometric(currentJoints,
                                        ctrlPoints,
                                        moveTime)){
  DisplayError(motionPlanner.GetError());
  return 0;
}

do{
  if (!motionPlanner.GetJointPosition(currentJoints,
                                      jointVel,
                                      state)){
    DisplayError(motionPlanner.GetError());
    break;
  }
  SetJoints(currentJoints);

}while(state != TrajectoryGenerator::TrajectoryComplete);
```

Figure 7.14. Example MP Code

7.2. DEMONSTRATION

7.2.1. Introduction

The first part of this chapter described the development and implementation of a new method for defining spatial motions. This work had three basic steps. First, the local geometric properties of curvature and torsion were studied in detail to provide a better

physical understanding. Then, a method to convert these properties into parametric constraints that can be used to define a spatial curve was developed. Finally, the resulting curve generation method was implemented inside an OSCAR-based Motion Planner. Now, this section will demonstrate how these methods can all be used to plan spatial motions for manipulators. First, a simple simulation environment will be described. Then, it will be shown how (and why) the various geometric constraints (i.e. curve parameters) can be modified to affect the local geometry at key frames of interest. Then, an example of how to string these together into an overall path plan will be shown. Finally, the affects of different velocity profiles on the motion of the manipulator will be explored.

7.2.2. Simulation Environment

The simulation environment used for this demonstration is shown in Figure 7.15. This shows a 7-DOF Mitsubish PA-10 manipulator and four key frames of interest. These frames are:

1. Initial manipulator configuration (i.e. home position)
2. A frame attached to a spherical object
3. A frame attached to a tray pick-up point
4. Final manipulator configuration (before returning to Position 1)

Thus, frames 2 and 3 are attached to specific objects (geometries) while frames 1 and 4 are not. The position and orientation of these frames must be defined carefully as the geometric constraints developed in this work always work relative to these frames. For example, the normal vector at frame 2 is defined to be pointing into the sphere to allow curvature to control the motion along the sphere's surface. Similarly, the tangent vector at frame 3 is defined to be pointing into the tray pick-up point to allow a cusp to be defined around this tangent. It should be noted that the positioning of these frames would most likely be done by the engineer designing the various components and would be

provided to the operator. The positions and orientations of these frames are shown in Table 7.8.

Frame	p	$\hat{\mathbf{T}}$	$\hat{\mathbf{N}}$	$\hat{\mathbf{B}}$
1	[0.106, 0.0, 0.975]	[1, 0, 0]	[0, −1, 0]	[0, −1, 0]
2	[0.4, 0.15, 0.5]	[0, −1, 0]	[0, 0, −1]	[1, 0, 0]
3	[0.5, −0.3, 0.525]	[1, 0, 0]	[0, 1, 0]	[0, 0, 1]
4	[0.2, −0.5, 0.6]	[0, −1, 0]	[0, 0, −1]	[1, 0, 0]

Table 7.8. Key Frame Positions and Orientations

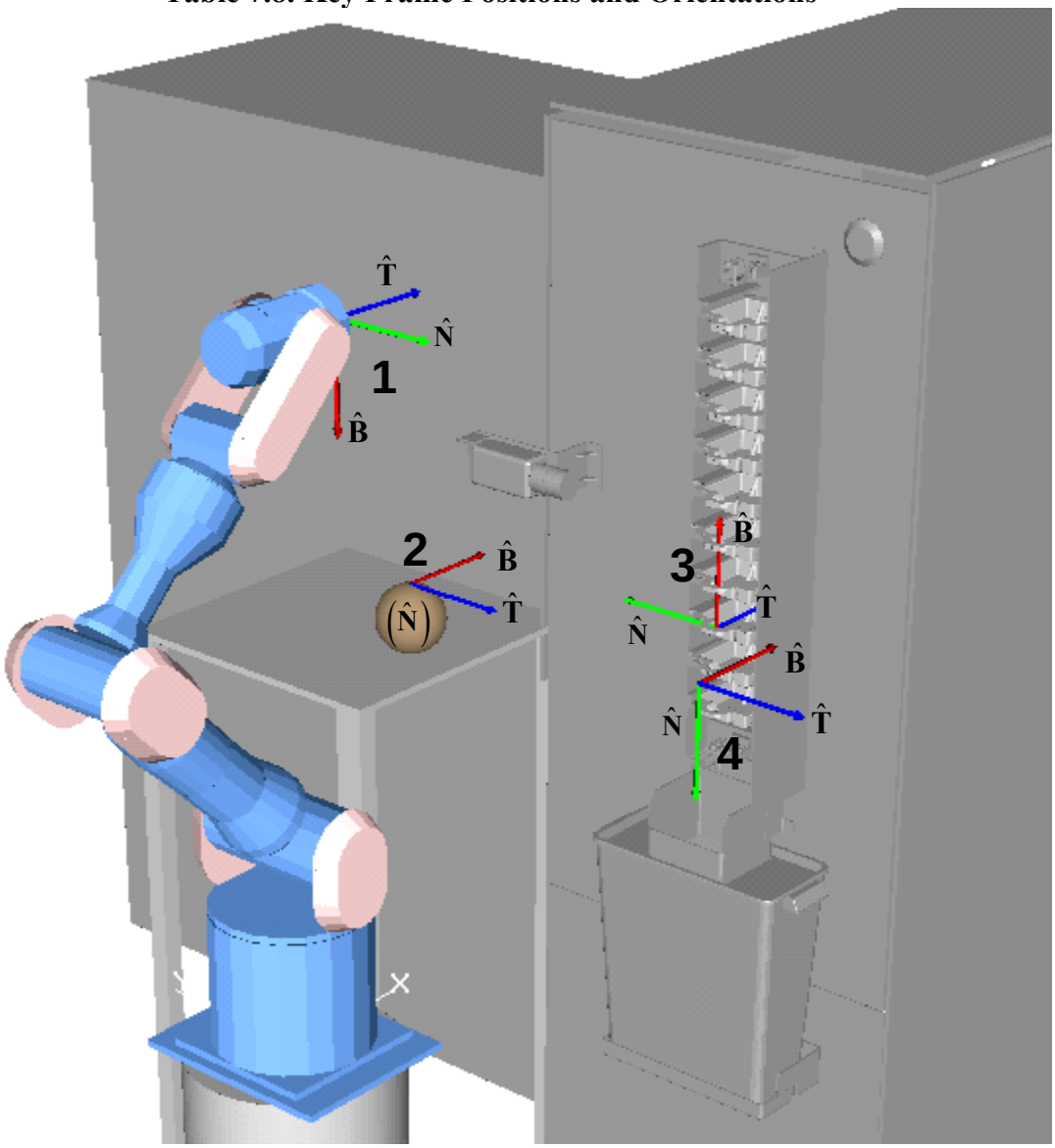


Figure 7.15. Simulation Environment

7.2.3. Tangent Scaling

As mentioned in Section 5.2 and 6.3.2.1, the simplest parameter that can be used to affect the shape of the curve is the tangent scale. This parameter basically shows that the first-order properties $\left(\frac{d\mathbf{p}}{du}\right)$ can be multiplied by any positive scalar while still maintaining the same geometric tangent. Thus, the first order parametric constraints are defined based on this value as $\frac{d\mathbf{p}}{du} = scale \times \hat{\mathbf{T}}(\mathrm{u})$. For example, suppose all of the higher-order geometric parameters were set to zero $(\kappa = \tau = \kappa' = \tau' = 0)$ and only the tangent scale was changed. In this case, the first order parametric constraints for three different values could be calculated as shown in Table 7.9 with the higher-order parametric constraints all being set to zero. The resulting family of curves is shown in Figure 7.16 for these three values. From this plot, the curve is moving relative to each frame only in the direction of the unit tangent vector. This is expected, because the higher-order geometric properties are all zero. This parameter can be useful in changing the overall shape of a curve without affecting the local higher-order geometric constraints at the frames. For the rest of this demonstration, a tangent scale of 0.25 is used to highlight the other parameters.

	scale	$\frac{d\mathbf{p}_1}{du}$	$\frac{d\mathbf{p}_2}{du}$	$\frac{d\mathbf{p}_3}{du}$	$\frac{d\mathbf{p}_4}{du}$
Path A	0.25	[0.25, 0.0, 0.0]	[0.0, -0.25, 0.0]	[0.25, 0.0, 0.0]	[0.0, -0.25, 0.0]
Path B	0.50	[0.50, 0.0, 0.0]	[0.0, -0.50, 0.0]	[0.50, 0.0, 0.0]	[0.0, -0.50, 0.0]
Path C	0.75	[0.75, 0.0, 0.0]	[0.0, -0.75, 0.0]	[0.75, 0.0, 0.0]	[0.0, -0.75, 0.0]

Table 7.9. Parametric Constraints with Varying Tangent Scale

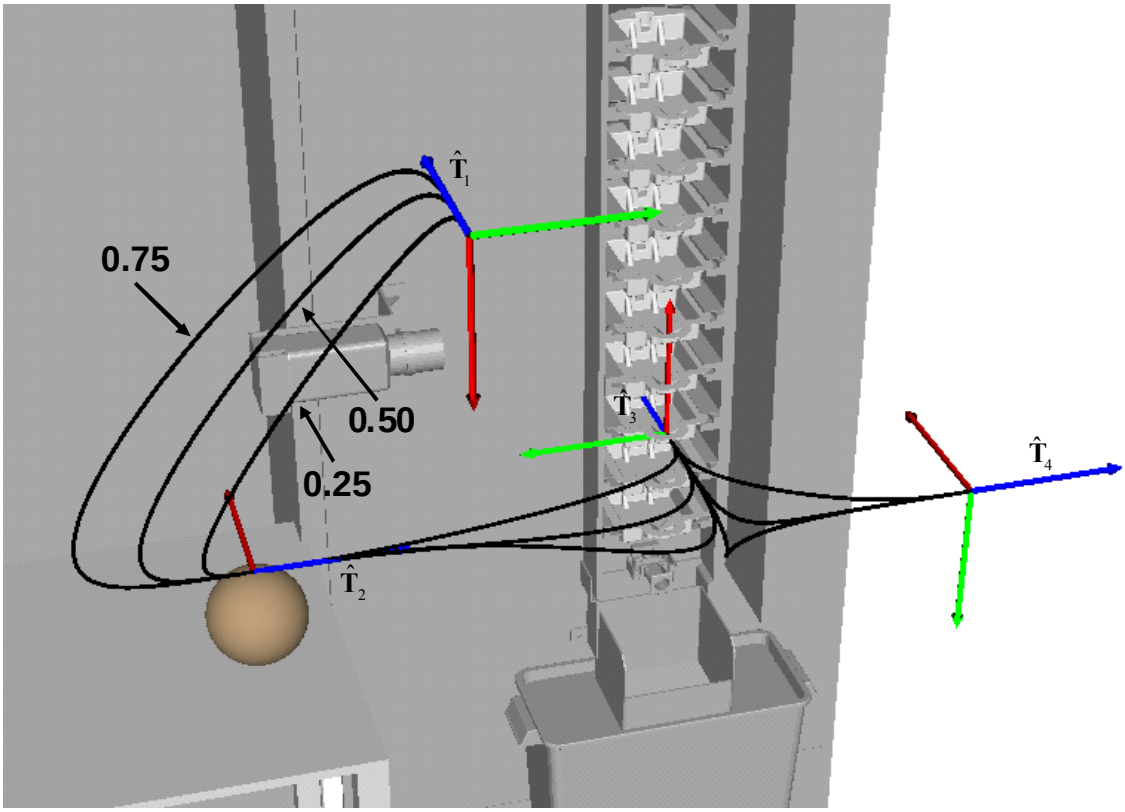


Figure 7.16. Effect of Varying Tangent Scale

One other important physical property of this parameter is that it is very dependent on the scale of the environment. For example, the points and curves defined in this demonstration are in meters and thus relatively small (i.e. < 1.0). However, if these points were instead defined in millimeters, these points would be much larger (up to ~1000). In this case, the tangent scale would need to be increased by a magnitude of 1000 as well in order to provide the same relative motion. Thus, the order of magnitude used for varying this parameter will be very dependant on the specific environment. Now, the next sections will show how the higher-order geometric parameters can be used to define the local shape around the desired frames.

7.2.4. Curvature

The simplest geometric constraint described in this research is curvature, κ. Curvature represents the reciprocal of the local radius of curvature. Thus, increasing curvature will increase the bending around the desired frame of interest. Curvature defines the second-order parametric constraint by the relationship $\frac{d^2\mathbf{p}}{du^2} = \left\|\frac{d\mathbf{p}}{du}\right\|^2 \kappa\hat{\mathbf{N}}$ (see Section 5.3). For example, Table 7.10 shows the parametric constraints calculated for three different values of curvature, and Figure 7.17 shows the resulting curves around frame 2. This clearly shows that increasing the curvature value increases the bending in the curve. Locally, this bending will occur within the osculating plane and around the unit normal vector. Thus, the placement of the frame is important. Here, the unit normal is simply defined to be normal to the surface of the object facing inside as can be seen in Figure 7.15. As the curvature increases, the curve appears to more closely match the shape of the sphere. However, if the curvature value gets too high (greater than 1/*r*), the curve will bend at a sharper rate locally than the sphere and cause undesired collisions. Thus, the maximum value of curvature allowed at an interaction point is constrained by the geometry of the part.

	κ	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{dz_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$	$\frac{d^2z_2}{du^2}$
Path A	1.0	0.0	-0.25	0.0	0.0	0.0	-0.0625
Path B	10.0	0.0	-0.25	0.0	0.0	0.0	-0.625
Path C	20.0	0.0	-0.25	0.0	0.0	0.0	-1.25

Table 7.10. Parametric Constraints with Varying Curvature at Frame 2

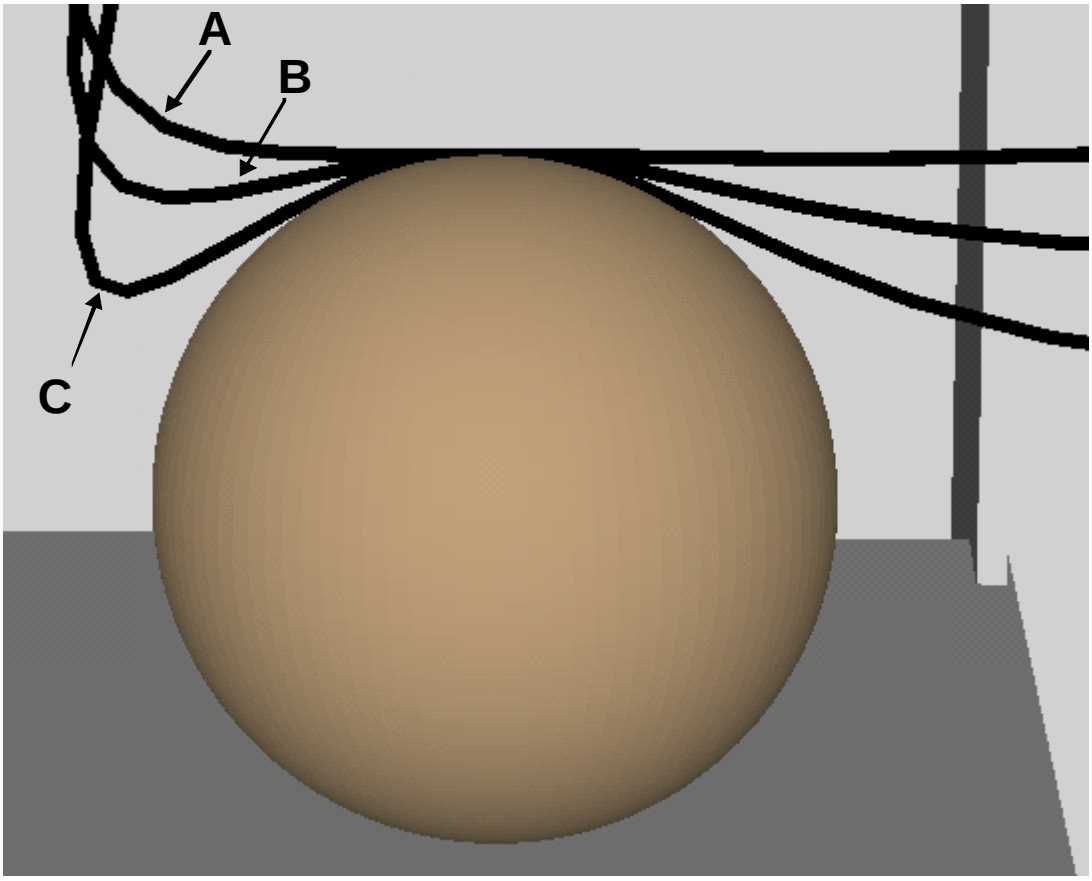


Figure 7.17. Varying Curvature at Frame 2

A similar analysis can be done at frame 2. Table 7.11 shows the calculated parametric constraints for three different values of curvature at a cusp, and Figure 7.18 shows the resulting curves. Once again, the larger values of curvature represent a larger bending. At a point such as this where an insertion/extraction task is taking place, it is probably desirable to have a lower value for curvature to create a "straighter" approach to the tray pickup point. For a free space task (such as picking an item off a conveyor), a larger curvature value may be useful. It should be noted that we use the geometric interpretation of a cusp here (unit tangent vector inverting) instead of the mathematical interpretation ($\kappa = \infty$). This is done because an infinite curvature leads to undefined or uncontrollable parametric constraints.

	κ	$\frac{dx_3}{du}$	$\frac{dy_3}{du}$	$\frac{dz_3}{du}$	$\frac{d^2x_3}{du^2}$	$\frac{d^2y_3}{du^2}$	$\frac{d^2z_3}{du^2}$
Path A	1.0	0.25	0.0	0.0	0.0	0.0625	0.0
Path B	10.0	0.25	0.0	0.0	0.0	0.625	0.0
Path C	20.0	0.25	0.0	0.0	0.0	1.25	0.0

Table 7.11. Parametric Constraints with Varying Curvature at Frame 3

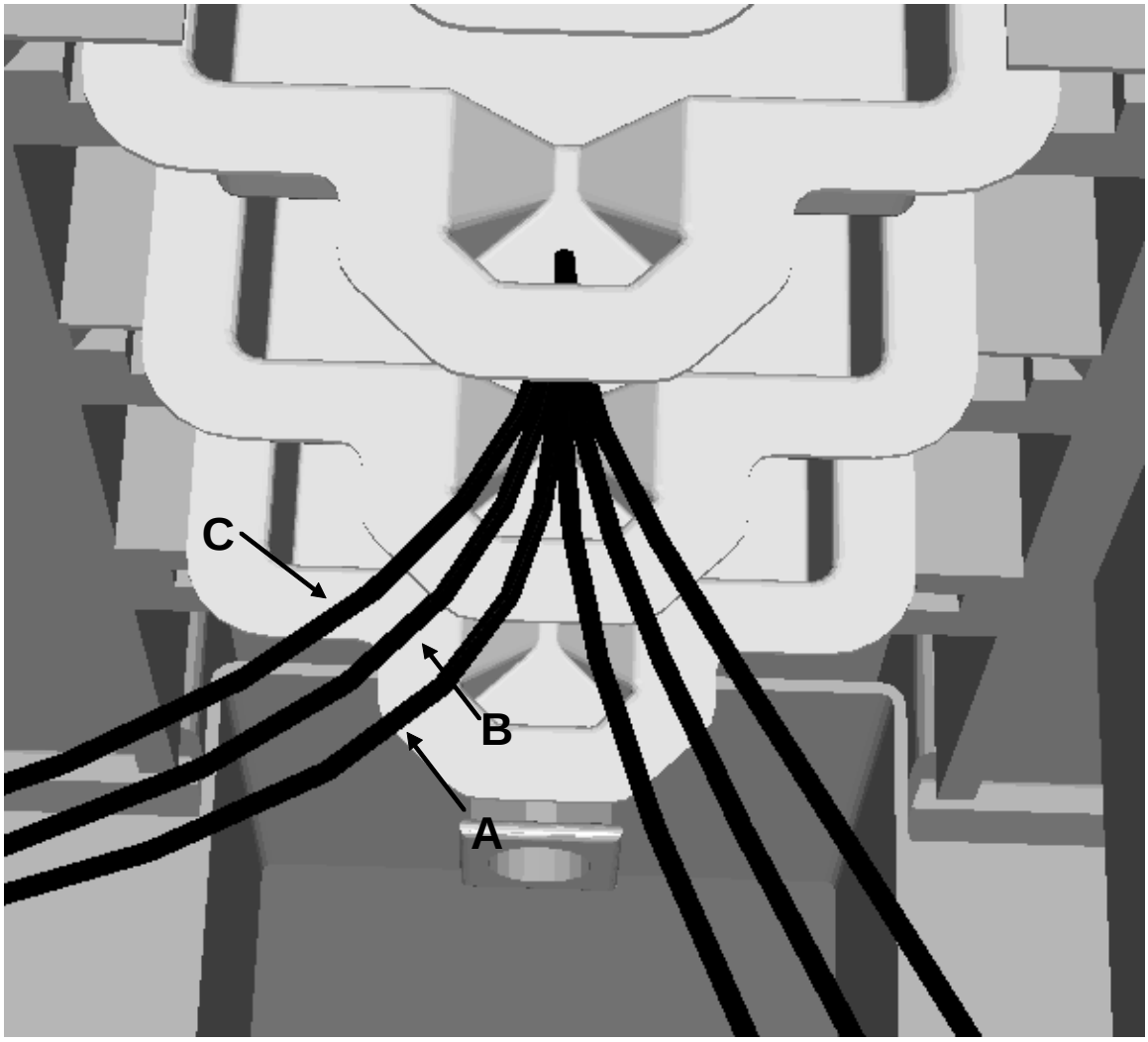


Figure 7.18. Varying Curvature at Frame 3

7.2.5. Torsion

The next geometric constraint studied in this work is torsion. As mentioned in the earlier studies of these properties, curvature basically defines the motion in the osculating plane while torsion defines the motion out of the plane (i.e. the bi-normal direction). Also, as mentioned earlier, torsion has less of an effect on the shape of the curve than curvature for similar numerical values. Thus, higher values of torsion are required to have noticeable effects. Torsion is used to define the third-order parametric constraints. To do this, the parametric equation is written as shown in Equation 7.16.

$$\tau(u) = \frac{x'''\left(y'z'' - y''z'\right) + y'''\left(x''z' - x'z''\right) + z'''\left(x'y'' - x''y'\right)}{\left(y'z'' - y''z'\right)^2 + \left(x''z' - x'z''\right)^2 + \left(x'y'' - x''y'\right)^2} \qquad \textbf{7.16}$$

Because the first and second-order parametric constraints have already been defined, these values can be considered constants in this equation. Then, solving for the third-order parametric constraints becomes equivalent to solving the simple linear system shown in Equation 7.17 (see Section 5.4 for full derivation).

$$\tau(u) = a_0 x''' + b_0 y''' + c_0 z''' \qquad \textbf{7.17}$$

For example, Table 7.12 shows the parametric constraints for varying torsions at frame 2, and Figure 7.19 shows the resulting curves. Another thing to note is that, unlike curvature which is always positive, torsion is a signed value. Thus, the positive values of torsion cause the curve to bend in the direction of the bi-normal vector, and the negative values cause it to bend in the opposite direction of the bi-normal vector. Increasing the magnitude of torsion also increases the amount of bending in this direction.

	κ	τ	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{dz_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$	$\frac{d^2z_2}{du^2}$	$\frac{d^3x_2}{du^3}$	$\frac{d^3y_2}{du^3}$	$\frac{d^3z_2}{du^3}$
Path A	20	50	0.0	-0.25	0.0	0.0	0.0	-1.25	15.625	0.0	0.0
Path B	20	100	0.0	-0.25	0.0	0.0	0.0	-1.25	31.25	0.0	0.0
Path C	20	-50	0.0	-0.25	0.0	0.0	0.0	-1.25	-15.625	0.0	0.0
Path D	20	-100	0.0	-0.25	0.0	0.0	0.0	-1.25	-31.25	0.0	0.0

Table 7.12. Parametric Constraints for Varying Torsion at Frame 2

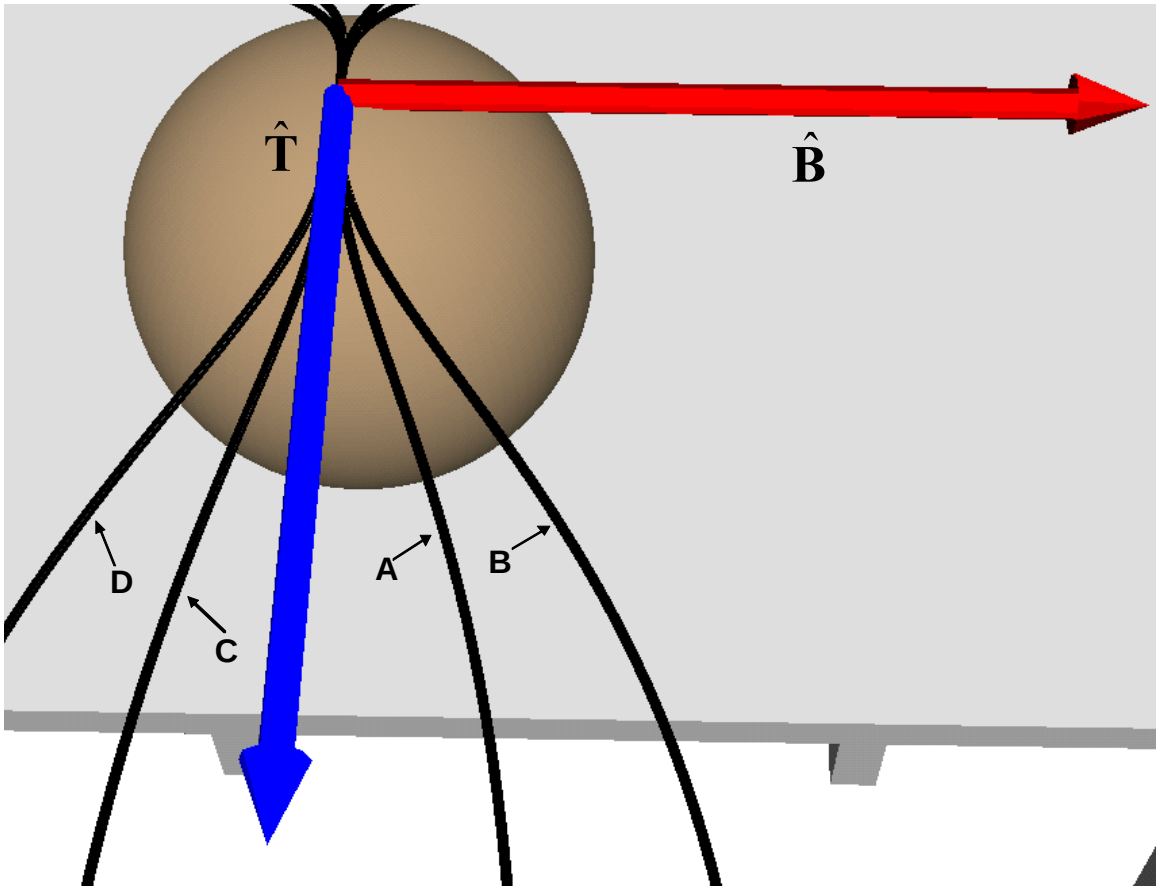


Figure 7.19. Varying Torsion at Way Point 1 with $\kappa = 20$

Now, it is useful to look at this family of curves from several points of view to fully understand the physical effects of these parameters on the curve. Figure 7.20 shows the same family of curves shown above from a slightly different angle. From this angle, it can be seen that these curves also have motion outside of the plane defined by the tangent

and bi-normal vectors (i.e. the view shown in Figure 7.19). This makes sense, because these curves also have values for curvature[14] which should generate some motion in the osculating plane (the tangent-normal plane).

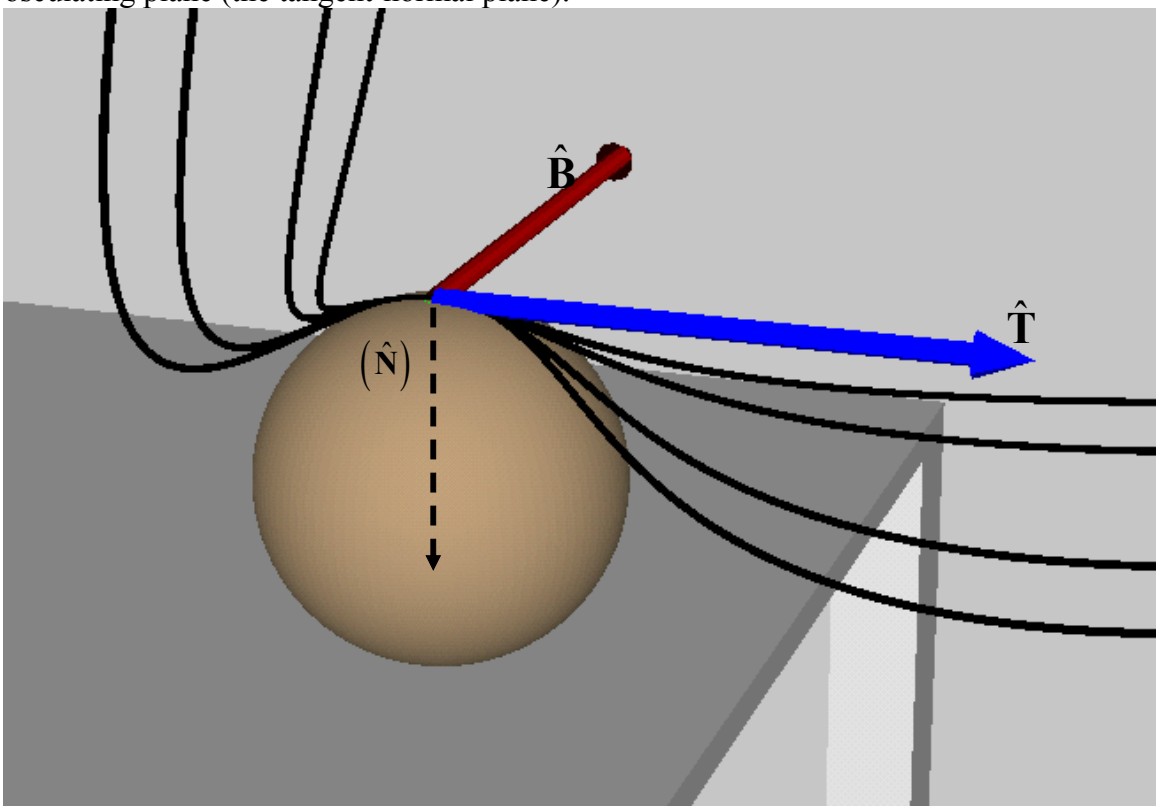


Figure 7.20. Varying Torsion at Frame 2 from a different perspective

Figure 7.21 shows this family of curves as viewed along the bi-normal vector (i.e. the osculating plane). This shows that locally the behavior of these curves is identical in this plane. This follows since as the curvature fully defines the motion in this plane, and all of these curves have identical values of curvature.

[14] In fact, these curves must have positive values of curvature, because torsion is undefined for zero curvature.

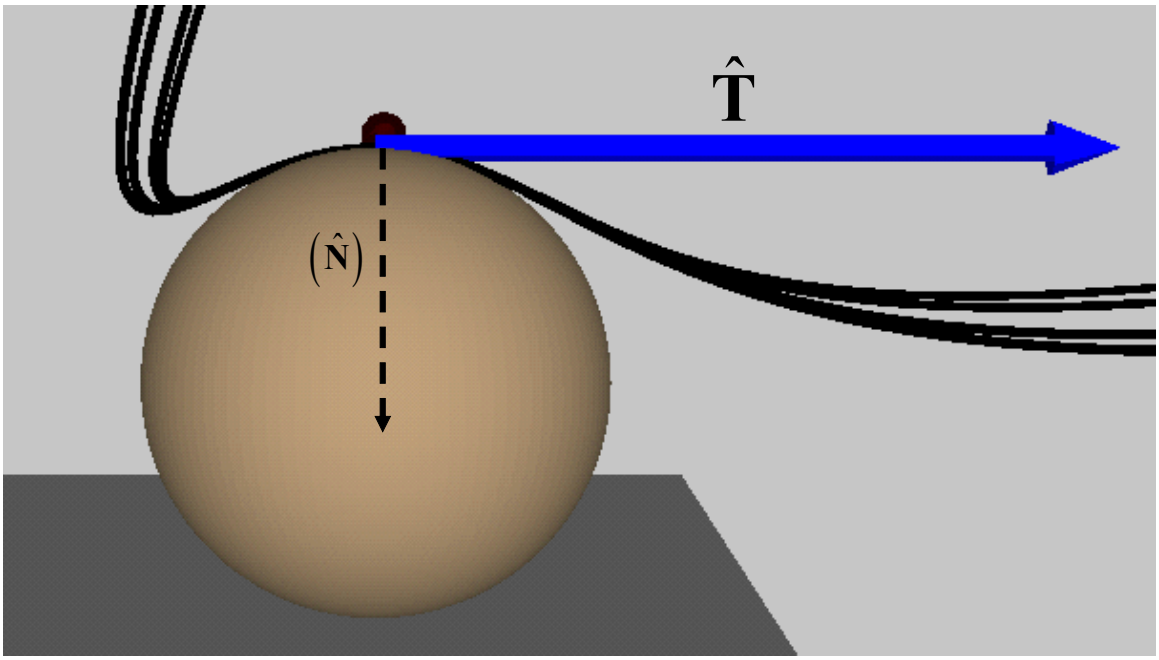


Figure 7.21. Varying Torsion from perspective of Osculating Plane

One other physical property of torsion is that the motion in the bi-normal direction is affected both by curvature and torsion. Thus, by lowering the value of curvature at a point, the relative effects of torsion also become smaller. For example, consider the same values of torsion as before with smaller values for curvature as shown in Table 7.13. The resulting family of curves is shown in Figure 7.22. Thus, if a curve is being designed by simply modifying these properties, a desirable value of curvature should be found first.

	κ	τ	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{dz_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$	$\frac{d^2z_2}{du^2}$	$\frac{d^3x_2}{du^3}$	$\frac{d^3y_2}{du^3}$	$\frac{d^3z_2}{du^3}$
Path A	5	50	0.0	-0.25	0.0	0.0	0.0	-.3125	3.9063	0.0	0.0
Path B	5	100	0.0	-0.25	0.0	0.0	0.0	-.3125	7.8125	0.0	0.0
Path C	5	-50	0.0	-0.25	0.0	0.0	0.0	-.3125	-3.9063	0.0	0.0
Path D	5	-100	0.0	-0.25	0.0	0.0	0.0	-.3125	-7.8125	0.0	0.0

Table 7.13. Parametric Constraints for Varying Torsion at Frame 2

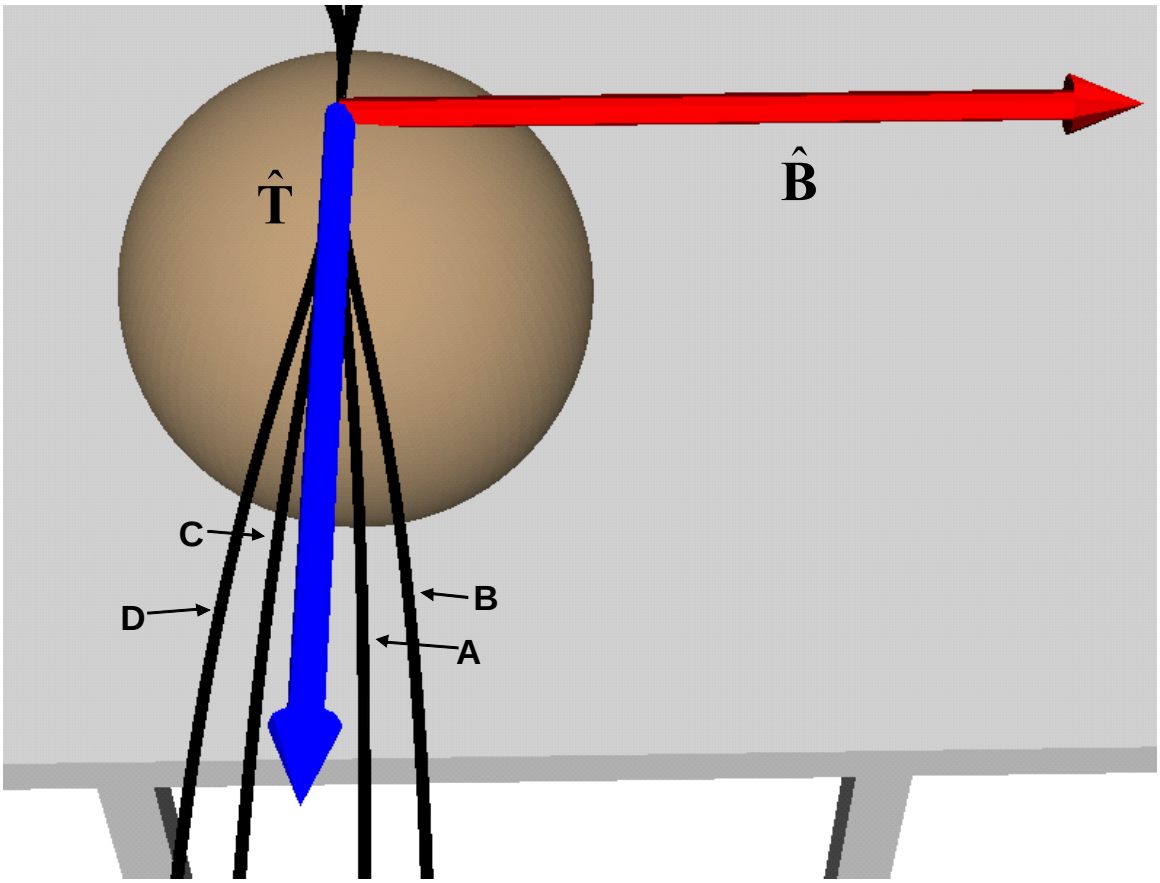


Figure 7.22. Varying Torsion at Way Point 1 with $\kappa = 5$

As before, we can also view this family of curves along the bi-normal vector to examine the motion in the osculating plane. This is shown in Figure 7.23. As expected, this family of curves has a much smaller motion in the osculating plane than the curves shown in Figure 7.21 due to the smaller values of curvature. However, the motion relative to osculating plane is still identical locally.

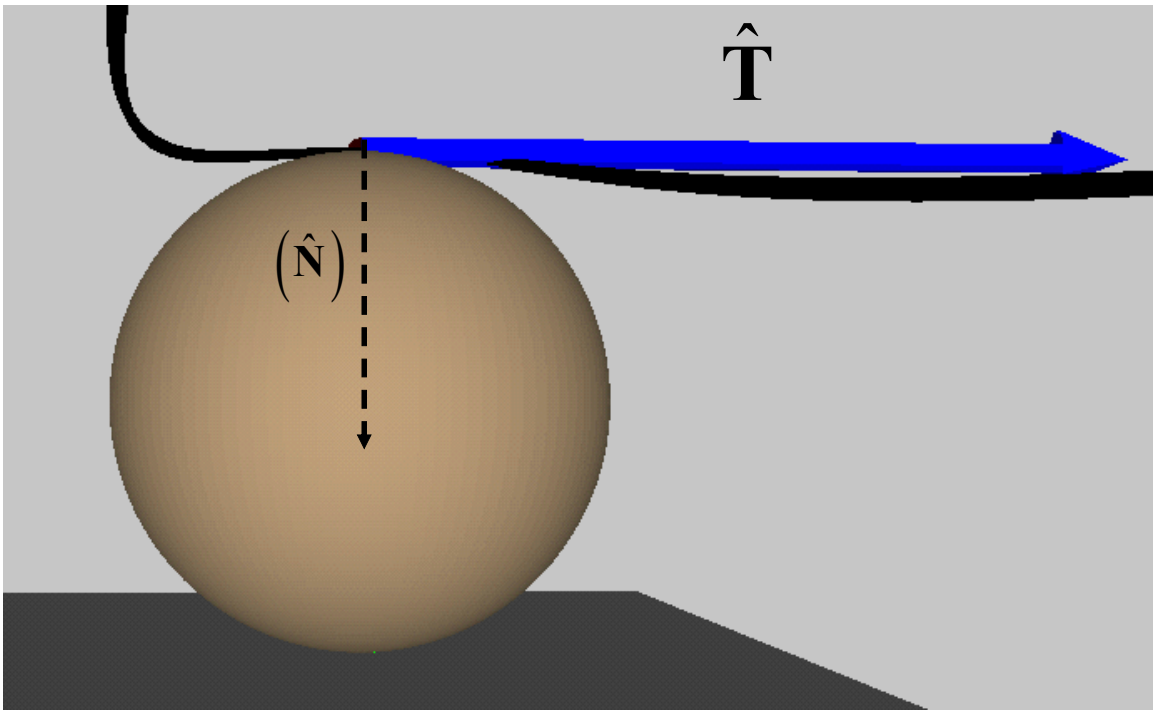


Figure 7.23. Perspective on Osculating Plane

These physical results can be further explained using the relationships shown in Equation 7.18 (first presented in Section 2.4.2). These relationships approximate the local motion of a curve in terms of curvature and torsion where the x_1, x_2, and x_3 axes correspond to the Tangent, Normal, and Bi-normal directions [26]. These equations show that curvature is the dominant factor in motion along the normal vector (in the osculating plane) while both curvature and torsion affect the motion in the bi-normal direction (out of the osculating plane). Also, because these relationships are only valid in a local sense, the values of s would be very small for the relevant local region. Thus, s^2 would be much larger than s^3. This further explains why torsion values must be higher than curvature values to have much effect on the local shape of a curve.

$$\begin{aligned} x_1(0) &= s + \cdots \\ x_2(0) &= \frac{\kappa}{2} s^2 + \cdots \\ x_3(0) &= \frac{\kappa\tau}{6} s^3 + \cdots \end{aligned} \qquad \textbf{7.18}$$

A similar family of curves with varying torsion is shown for the second frame in Figure 7.24. The parametric constraints for these curves are shown in Table 7.14. As before, positive/negative torsion is used to define the motion approaching and leaving the cusp point. Here, the motion is relative to the *z* axis as that is the direction the bi-normal vector is pointing (see Table 7.8).

	κ	τ	$\frac{dx_3}{du}$	$\frac{dy_3}{du}$	$\frac{dz_3}{du}$	$\frac{d^2x_3}{du^2}$	$\frac{d^2y_3}{du^2}$	$\frac{d^2z_3}{du^2}$	$\frac{d^3x_3}{du^3}$	$\frac{d^3y_3}{du^3}$	$\frac{d^3z_3}{du^3}$
Path A	20	50	0.25	0.0	0.0	0.0	1.25	0.0	0.0	0.0	15.625
Path B	20	100	0.25	0.0	0.0	0.0	1.25	0.0	0.0	0.0	31.25
Path C	20	-50	0.25	0.0	0.0	0.0	1.25	0.0	0.0	0.0	-15.625
Path D	20	-100	0.25	0.0	0.0	0.0	1.25	0.0	0.0	0.0	-31.25

Table 7.14. Parametric Constraints with Varying Torsion at Frame 3

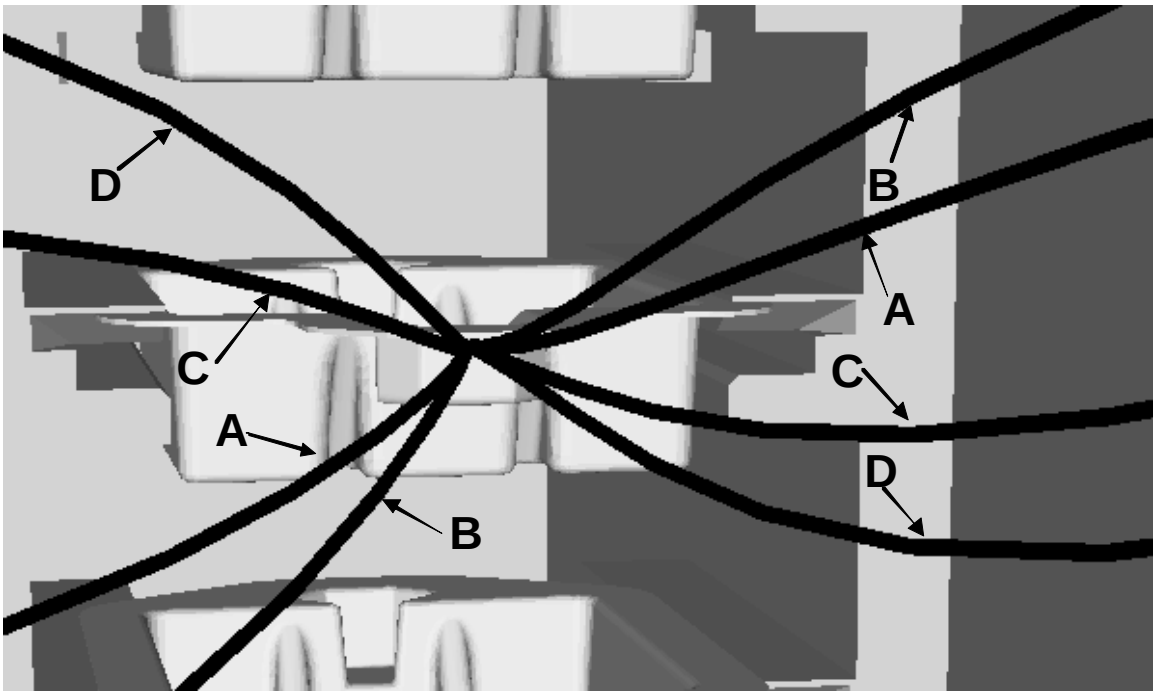


Figure 7.24. Varying Torsion at Way Point 1

7.2.6. Higher-Order Properties

The last two sections described the effects of curvature and torsion on the local geometry of a curve and showed how these parameters could be varied to produce familes of local curves. However, it was shown in Chapter 5 how to formulate parametric constraints for $\frac{d\kappa}{du}$ and $\frac{d\tau}{du}$ as well. While these parameters can also affect the shape of the local geometry of a curve, the influences of curvature and torsion are much larger (relatively). Thus, larger magnitudes must be provided for these constraints to have much effect on the local geometry of the curve. For example, Table 7.15 shows the parametric constraints for three differing values of $\frac{d\kappa}{du}$ (see Section 5.4 for derivation), and Figure 7.25 shows a family of curves created by varying this parameter. From this plot, it can be seen that these curves do not vary greatly despite the large magnitudes for $\frac{d\kappa}{du}$ (0, 200, and 400).

	κ	κ'	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{dz_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$	$\frac{d^2z_2}{du^2}$	$\frac{d^3x_2}{du^3}$	$\frac{d^3y_2}{du^3}$	$\frac{d^3z_2}{du^3}$
Path A	10	0	0.0	-0.25	0.0	0.0	0.0	-0.625	0.0	0.0	0.0
Path B	10	200	0.0	-0.25	0.0	0.0	0.0	-0.625	0.0	0.0	-12.50
Path C	10	400	0.0	-0.25	0.0	0.0	0.0	-0.625	0.0	0.0	-25.00

Table 7.15. Parametric Constraints for Varying Derivative of Curvature at Frame 2

The best way to understand the physical meaning of derivative of curvature is to think of its relationship to the actual curvature value (i.e. reciprocal of local radius of curvature). The derivative of curvature represents how fast the curvature value is changing at a particular point. Thus, if a zero value is provided, this represents a local minima or maxima in the curvature profile at the point. This will result in a better match for this constraint at the given point. A large value for $\frac{d\kappa}{du}$ means that the curvature is rapidly changing at the given point. For example, a large positive value for this parameter

means that the curvature is rapidly increasing at the desired point. This means that the curvature approaching the point will be much smaller than the curvature leaving the point. This can be seen in Path C in Figure 7.25 ($\frac{d\kappa}{du} = 400$). This curve approaches the point with a fairly straight trajectory and leaves with sharper bending. For a value of zero, the bending is symmetric around the specified point.

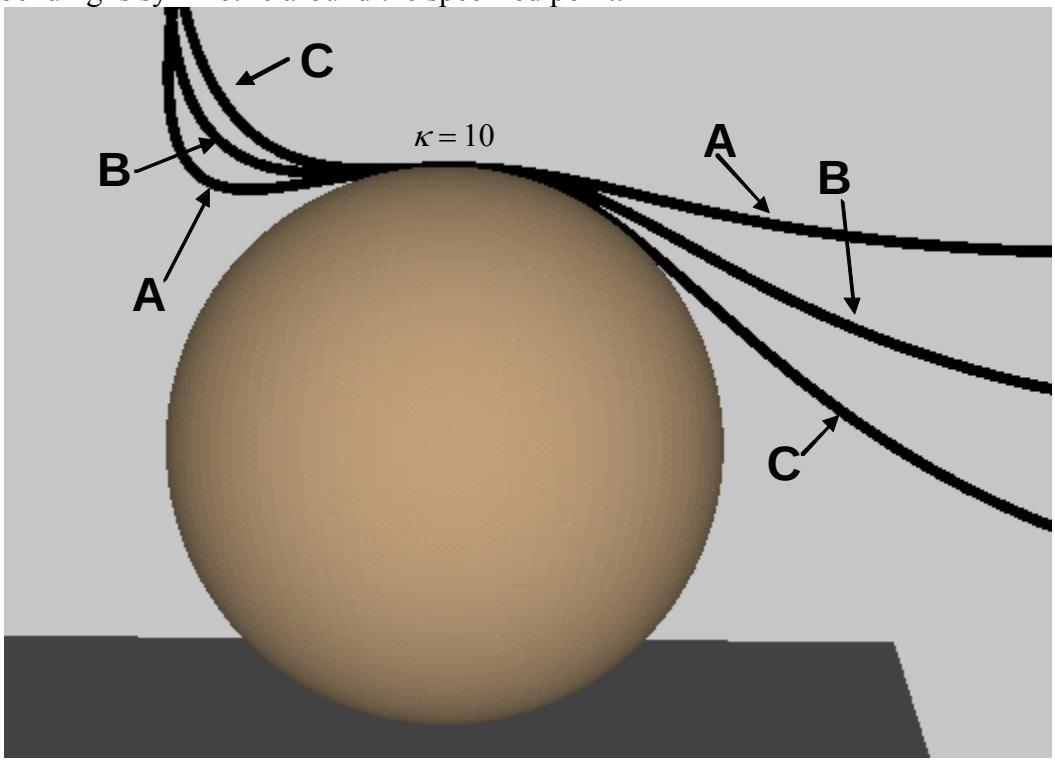


Figure 7.25. Varying Derivative of Curvature

Similarly, a good way to understand the physical meaning of $\frac{d\tau}{du}$ is to relate it to torsion. Derivative of torsion represents how fast the value of torsion is changing at some particular point. Thus, for example, a large positive value of $\frac{d\tau}{du}$ would mean that the torsion approaching the point is much smaller than the torsion leaving the point. This was used along with a constraint of $\tau = 0$ to create spatial saddle points in Section 4.4.3. The concept is that by setting $\tau = 0$ and $\frac{d\tau}{du}$ to some large positive value the torsion approaching the point will be negative and leaving the point will be positive. Thus, the

curve will switch directions in its motion relative to the bi-normal vector. The opposite result will occur when specifying a large negative value of $\frac{d\tau}{du}$.

For example, consider the geometric/parametric constraints shown in Table 7.16 (see Section 5.5 for full derivation). This shows three path specifications with $\tau = 0$ and varying values for $\frac{d\tau}{du}$ (with $\kappa = 10$ and $\kappa' = 0$). As with torsion relative to curvature, the derivative of torsion requires larger values relative to the derivative of curvature to have noticeable effects on the geometry of the curve. The resulting family of curves is shown in Figure 7.26. As expected, the larger values of $\frac{d\tau}{du}$ lead to a sharper bend around the bi-normal vector. Thus, physically, this parameter is useful to describing motions where the curve needs to change directions relative to the bi-normal vector at a particular point.

	Path A	**Path B**	**Path C**
κ	10	10	10
κ'	0	0	0
τ	0	0	0
τ'	1000	5000	10000
$\frac{d\mathbf{p}}{du}$	[0.0, -0.25, 0.0]	[0.0, -0.25, 0.0]	[0.0, -0.25, 0.0]
$\frac{d^2\mathbf{p}}{du^2}$	[0.0, 0.0, -0.625]	[0.0, 0.0, -0.625]	[0.0, 0.0, -0.625]
$\frac{d^3\mathbf{p}}{du^3}$	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]
$\frac{d^4\mathbf{p}}{du^4}$	[156.25, 0.0, 0.0]	[781.25, 0.0, 0.0]	[1562.5, 0.0, 0.0]

Table 7.16. Parametric Constraints with Varying Derivative of Torsion at Frame 2

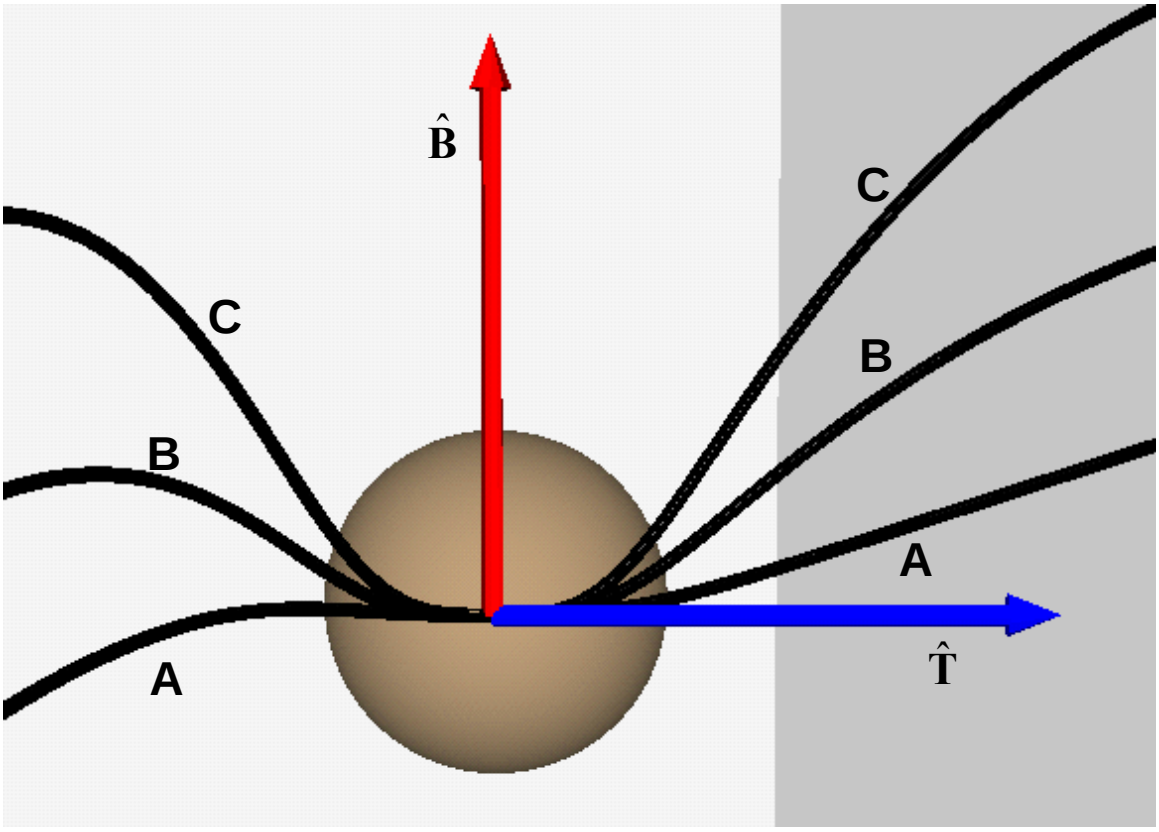


Figure 7.26. Varying Derivative of Torsion

7.2.7. Summary of Geometry Parameters

In the last sections, each geometric parameter developed in this research was discussed, and the local effects of these parameters were examined. These parameters represent degrees of freedom that a user or operator can utilize in designing the geometry of spatial curves. These parameters can be used either to interactively modify the geometry of a curve or to define a specific physical constraint on the curve (e.g. matching the radius of curvature of a spherical object). Table 7.17 provides a summary of these geometric parameters as discussed in the previous sections.

Table 7.17. Summary of Geometric Parameters

Geometric Constraint	Symbol	Description	Example Results
Tangent Scale	*scale*	This parameter controls the "bias" towards the unit tangent vector. Physically, it is important for adjusting for the scale of the environment (e.g. meters vs. millimeters, etc). It also allows for some interactive control over the global shape of the spatial curve (without affecting the local geometric parameters).	• Table 7.9 shows the effect on the first-order parametric constraints $\left(d\mathbf{p}/du\right)$ for three different values for the tangent scale (0.25, 0.5, 0.75) • Figure 7.16 illustrates visually how this parameter influences the local geometry along the unit tangent
Curvature	κ	Used to define the local reciprocal of radius of curvature. A zero value corresponds to a straight line while an infinite value corresponds to a discontinuity in the unit tangent vector. Interactively, this parameter can be used to control the local "bending" around a frame. Physically, this can be used to match the geometry of an object or task with a desired radius.	• Table 7.10 shows the influence of curvature on the second-order parametric constraints $\left(d^2\mathbf{p}/du^2\right)$ for varying values of curvature (1, 10, 20) at a spherical object. • Figure 7.17 provides a visual illustration of varying curvature at the spherical object. As the curvature increases, the curve follows the surface of the sphere more closely. • Table 7.11 shows the second-order parametric constraints $\left(d^2\mathbf{p}/du^2\right)$ of varying values of curvature (1, 10, 20) at a cusp point • Figure 7.18 demonstrates visually the effects of varying curvature at the cusp point. Smaller values of curvature provide a straighter approach to the point, and larger values create more open cusp.

Table 7.17 (cont.)

Torsion	τ	Used to control the motion in the direction of the bi-normal vector (i.e. motion out of the osculating plane). A positive value will make the curve locally move in the bi-normal direction and a negative value will make the curve locally move in the inverse of the bi-normal direction. Torsion can also be coupled with curvature to describe specific physical spatial shapes (e.g. helices).	• Table 7.12 shows the relationships between torsion and the third-order parametric constraints $\left(d^3\mathbf{p} / du^3 \right)$ for four values of torsion (50, 100, -50, -100) with a value of $\kappa = 20$. These numeric values must be large relative to curvature to provide similar geometric effect. • Figure 7.19 illustrates visually the effect of this parameter on motion in the bi-normal direction. As the magnitudes get larger, the motion in the bi-normal direction becomes more pronounced. Also, positive and negative values are used to define the motion in either the bi-normal or inverse bi-normal directions. • Figure 7.20 and Figure 7.21 show this family of curves from different points of view to further illustrate the local effect of torsion. This demonstrates that curvature controls motion relative to the osculating plane and torsion controls motion outside of the osculating plane. • Table 7.13 shows the relationships between torsion and the third-order parametric constraints $\left(d^3\mathbf{p} / du^3 \right)$ for four values of torsion (50, 100, -50, -100) with value of $\kappa = 5$. • Figure 7.22 shows the resulting family of curves. This plot demonstrates the coupling of curvature and torsion by showing a much smaller effect compared to the earlier result with $\kappa = 20$.

Table 7.17 (cont.)

			• Table 7.14 and Figure 7.24 show the parametric constraints and resulting family of curves for varying torsion values (50, 100, -50, -100) at a cusp point. As before, this plot shows that torsion has an effect on the local motion in the bi-normal direction.
Derivative of Curvature	κ'	Controls the rate of change of the curvature around a point. This can be useful for altering the behavior of the curvature around a specific frame. For example, a large positive value will make the curve "flatter" when approaching a point than leaving the point. A zero value will force a local minima or maxima in the curvature profile that will lead to a better match for the desired curvature.	• Table 7.15 demonstrates the influence of derivative of curvature on the 3rd order parametric constraints $\left(d^3\mathbf{p}/du^3\right)$ at a spherical object for three values (0,200,400). • Figure 7.25 shows the geometric effect of this parameter on the curve. With a zero value, the curve closely matches the curvature both approaching and leaving the specified point. For larger positive values, the bending leaving the point will be greater than approaching.
Derivative of Torsion	τ'	Controls the rate of change of the torsion around a point. This parameter can be used to change the direction of motion along the bi-normal vector. For example, a zero value for torsion and a large positive value for τ' will force the torsion to be negative when approaching a point and positive when leaving the point. A zero value will force a local minima or maxima in the torsion profile that will lead to a better match for the desired torsion.	• Table 7.16 demonstrates the effect of derivative of torsion on the 4th order parametric constraints $\left(d^4\mathbf{p}/du^4\right)$ for three values of τ' (1000, 5000, 10000). • Figure 7.26 shows a family of spatial saddles points generated by varying derivative of torsion. This shows that the motion of the curve in the bi-normal switches at the defined position.

7.2.8. Full Motion Specification

In the previous sections, it has been shown how the local geometric properties at each frame can be used to define the motion relative to that frame. In this section, it will be shown how these various local constraints can be blended together into an overall motion plan. For this example, the geometric constraints at the first and last frame are assumed to be zero (i.e. only a tangent specification) as these frames are sitting freely in space rather than being attached to a physical geometry. Then, the curvature and torsion are varied at frames 2 and 3 to produce different overall motions. Derivative of curvature (κ') and torsion (τ') are also set to zero in this demonstration as the local effects of these parameters are difficult to visualize relative to the overall motion. The geometric constraints used for frames 2 and 3 are shown in Table 7.18.

	Frame 2		**Frame 3**	
	κ	τ	κ	τ
Path A	5	-50	1	0
Path B	10	-50	10	100
Path C	20	-50	10	-100

Table 7.18. Local Geometric Constraints

As before, these geometric constraints are converted into parametric constraints to define the overall curve geometry using the process described in Section 7.1.4. The calculated parameteric constraints for frame 2 and 3 are shown in Table 7.19 and Table 7.20, respectively.

	$\frac{dx_2}{du}$	$\frac{dy_2}{du}$	$\frac{dz_2}{du}$	$\frac{d^2x_2}{du^2}$	$\frac{d^2y_2}{du^2}$	$\frac{d^2z_2}{du^2}$	$\frac{d^3x_2}{du^3}$	$\frac{d^3y_2}{du^3}$	$\frac{d^3z_2}{du^3}$
Path A	0.0	-0.25	0.0	0.0	0.0	-0.3175	-3.9063	0.0	0.0
Path B	0.0	-0.25	0.0	0.0	0.0	-0.625	-7.8125	0.0	0.0
Path C	0.0	-0.25	0.0	0.0	0.0	-1.25	-15.625	0.0	0.0

Table 7.19. Parametric Constraints at Frame 2

	$\frac{dx_3}{du}$	$\frac{dy_3}{du}$	$\frac{dz_3}{du}$	$\frac{d^2x_3}{du^2}$	$\frac{d^2y_3}{du^2}$	$\frac{d^2z_3}{du^2}$	$\frac{d^3x_3}{du^3}$	$\frac{d^3y_3}{du^3}$	$\frac{d^3z_3}{du^3}$
Path A	0.25	0.0	0.0	0.0	0.0625	0.0	0.0	0.0	0.0
Path B	0.25	0.0	0.0	0.0	0.625	0.0	0.0	0.0	15.625
Path C	0.25	0.0	0.0	0.0	0. 625	0.0	0.0	0.0	-15.625

Table 7.20. Parametric Constraints at Frame 3

Now, these parametric constraints can be blended together to form the overall path geometry as shown in Figure 7.27. While the exact effects of the geometric constraints can be difficult to interpret, the relative effect of these constraints can be seen at each frame. For example, Path A appears to have the least bending at the Frame 2 while Path C has the sharpest bending. This is the expected behavior based on the assigned curvature values. At Frame 3, the effects of the assigned torsion values can be seen. Path B approaches and leaves the cusp point moving in the positive *z* direction (i.e. the bi-normal direction) due to its positive torsion value while the opposite behavior can be seen in Path C due to its negative torsion. Thus, the behavior of the local geometry around each frame is in line with the physical understanding of curvature and torsion outlined in the previous sections (7.2.1-7.2.7) of this report.

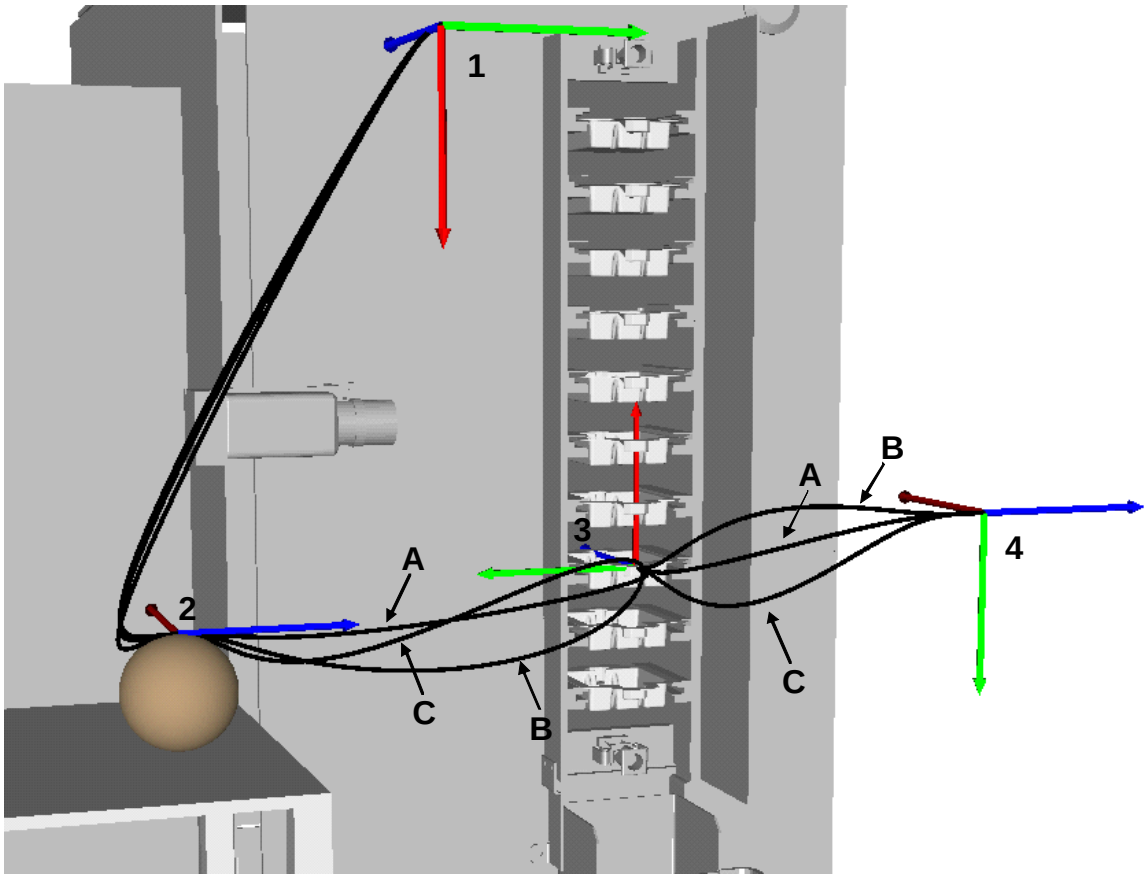


Figure 7.27. Overall Motion Trajectory

7.2.9. Motion Profiles

The previous section showed how to define the local geometry of the spatial path. Now, a spatial motion must be defined on top of this geometry $\left(u = f\left(t\right)\right)$ to provide a continuous path function through the desired points. As mentioned before, this involves defining a desired velocity profile and then using a 2nd order approximation to determine the correct value of u at every sampling period (100 hz in this example). This velocity profile, as defined in Section 6.3.2.4, is created by specifying a move time and then calculating the required spatial velocity and acceleration based on the total arc length of the path. This profile will also come to a stop (i.e. zero velocity) at any cusp points to prevent an instantaneous change in direction. Thus, a smooth velocity profile is internally

defined[15] in the MP and the resulting output velocities can be measured to see how well the approximation worked. For example, the output velocity profiles for the three paths specified in the previous section are shown in Figure 7.28 for a motion time for each of 10 seconds.

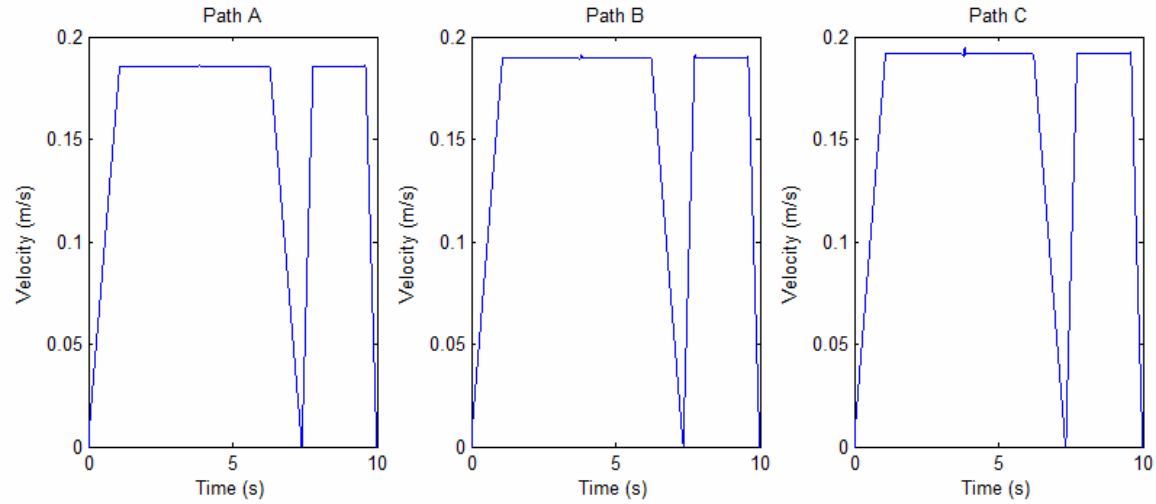


Figure 7.28. Velocity Profiles

These output (i.e. "measured") velocity profiles look almost identical from these plots. However, there are slight differences that can be difficult to see. First, the peak velocity increases slightly from Path A to Path C. This is because the overall arc length of the curve increases, and the velocity must also increase to complete the motion in the specified time. Also, though it is difficult to see in these graphs, there is some noise in the velocity profile. This error is at its largest at frame 2 where the curvature value is large. This computational error is summarized in Table 7.21 for each of the three motion specifications. This table shows that the maximum error will increase as the higher-order geometric properties get larger. It should be noted that this is not a problem at Frame 3, because the velocity around this point is small.

[15] This velocity profile used is not optimal as the main goal here is to measure how accurately the approximation method can follow a provided profile. A more in-depth evaluation and comparison of various motion profiles can be found in [33][38][50].

	Maximum Velocity	Maximum Velocity Error	% Velocity Error
Path A	0.1855	0.0007	0.3929
Path B	0.1899	0.0015	0.8135
Path C	0.1918	0.0026	1.3387

Table 7.21. Velocity Profile Errors

Now, the effect of changing the relative move time on the velocity profiles will be explored. Figure 7.29 shows the velocity profiles for Path C for three different move times: 5, 10, and 15 seconds.

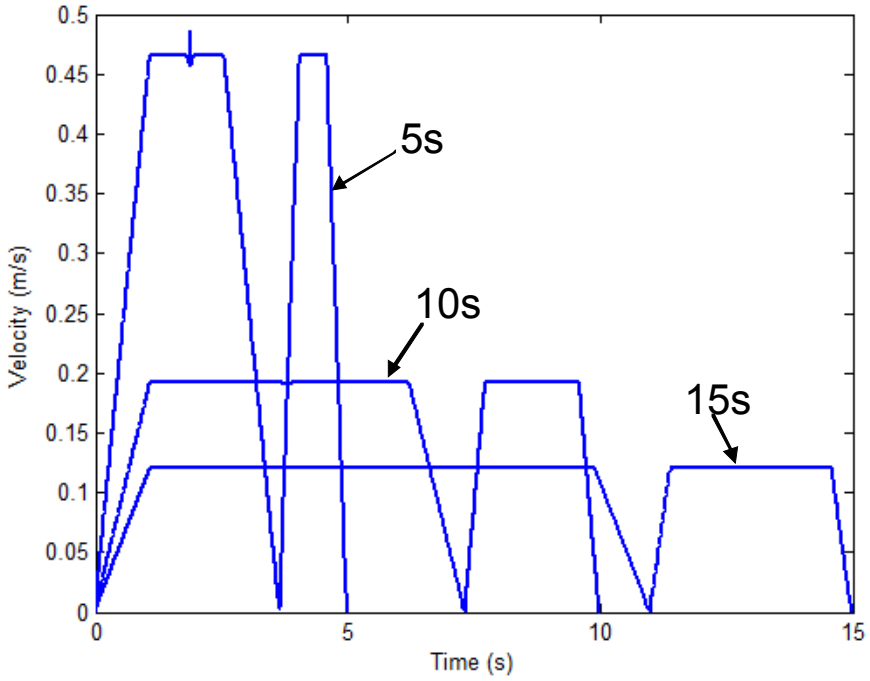


Figure 7.29. Velocity Profiles for Varying Move Times

As expected, the peak velocities decrease as the overall motion time becomes larger. Also, a noticeable error can be seen in the five second profile. This error occurs in the constant velocity portion of the first trapezoid and corresponds to the motion at frame 2 (where the curvature is large). A summary of these errors is shown in Table 7.22. This shows that errors will occur in the proposed velocity approximation method (i.e. $u = f(t)$) at points with large higher-order geometric properties (especially curvature) and/or large velocities. There are several potential ways to improve this approximation

method. First, a higher-order approximation could be formulated. Also, the errors may be lessened by increasing the sampling rate so that there is less distance between sampled points. However, the limitations of the actual physical system must be accounted for here as well.

Move Time	Maximum Velocity	Maximum Velocity Error	% Velocity Error
5s	0.4658	0.0197	4.2384
10s	0.1918	0.0026	1.3387
15s	0.1208	0.0008	0.6355

Table 7.22. Velocity Profile Errors for Varying Move Times

7.2.10. Conclusions

The above sections described how to use the geometric parameters studied in this work to create different local geometries for spatial curves. This method is similar to some of the methods described earlier in Chapter 3 in that it provides variable parameters that can be used to to change and adjust the shape of the spatial curve. However, in this work, these variable parameters (i.e. geometric constraints) have clear physical meaning that should be useful in defining physical tasks and interactions. These techniques have also been packaged into a robot-independent software architecture. This allows for the motion along a curve defined by geometric constraints to be easily programmed for a variety of physical systems and manipulators. The next section of this report will describe future improvements to the techniques developed in this work as well as describe some potential future application areas.

7.3. FUTURE WORK

In this section, several suggestions for future extension and application of this research will be presented. First, a number of improvements that can be made to the existing framework will be suggested. This involves the low-level curve generation

mathematics, rotational motion specification, and the actual software implementation. Then, a number of future applications and areas of potential research will be described.

7.3.1. Curve Generation Techniques

The main focus of this research has been on the understanding and definition of physical constraints for manipulator path planning. This started with a thorough study of curvature and torsion in Chapter 4, and then these geometric constraints were converted into parametric constraints in Chapter 5. While several curve generation techniques such as polynomials and trapezoids (see Appendix B for a further description) were briefly examined, curve generation schemes have not been a priority. This section will present several possibilities for future work in this area.

One of the main goals of most curve generation schemes is to keep the curve of as low order as possible to prevent unpredictable behavior. However, the methods developed in this work require parametric constraints to be defined up to the fourth order. Thus, by necessity, high order curves must be defined to meet these constraints. Along these lines, one of the simplest ways to improve this scheme would be to keep the order of the curve as low as possible. This can be easily done by only utilizing the required end constraints to define a curve. For example, if only a curvature constraint needs to be defined at an end point, parametric constraints need only be defined up to the second order. This simple improvement may produce better behavior between end constraints.

Another important area of future work is the application/comparison of different curve generation techniques. While this research focused on the generation of parametric end constraints for curves, some of the methods reviewed in Chapter 3 can now be used to meet these constraints. For example, consider the n^{th} order Bezier curve defined by Equation 7.19 where $\mathbf{b}_i$ represent a set of control points and $B_i^n(u)$ represent the Bezier basis functions.

$$\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(u), u \in [0,1] \qquad \textbf{7.19}$$

Now, the parametric derivatives of this curve at the end points can be easily calculated [10]. Equation 7.20 shows the first-order derivatives, and Equation 7.21 shows the second-order derivatives. As mentioned in Chapter 3, Bezier curves are often designed by interactively moving the control points ($\mathbf{b}_i$) to form the desired visual curve. However, these equations show that the same geometric based constraints developed in this work can be used to define the location of these control points. Thus, the same local phenomena can be described using Bezier curves. An example of this for varying curvature at a specific point is shown in Figure 7.30.

$$\frac{d\mathbf{p}(0)}{du} = n(\mathbf{b}_1 - \mathbf{b}_0)$$
$$\frac{d\mathbf{p}(1)}{du} = n(\mathbf{b}_n - \mathbf{b}_{n-1}) \qquad \textbf{7.20}$$

$$\frac{d^2\mathbf{p}(0)}{du^2} = n(n-1)(\mathbf{b}_0 - 2\mathbf{b}_1 + \mathbf{b}_2)$$
$$\frac{d^2\mathbf{p}(1)}{du^2} = n(n-1)(\mathbf{b}_{n-2} - 2\mathbf{b}_{n-1} + \mathbf{b}_n) \qquad \textbf{7.21}$$

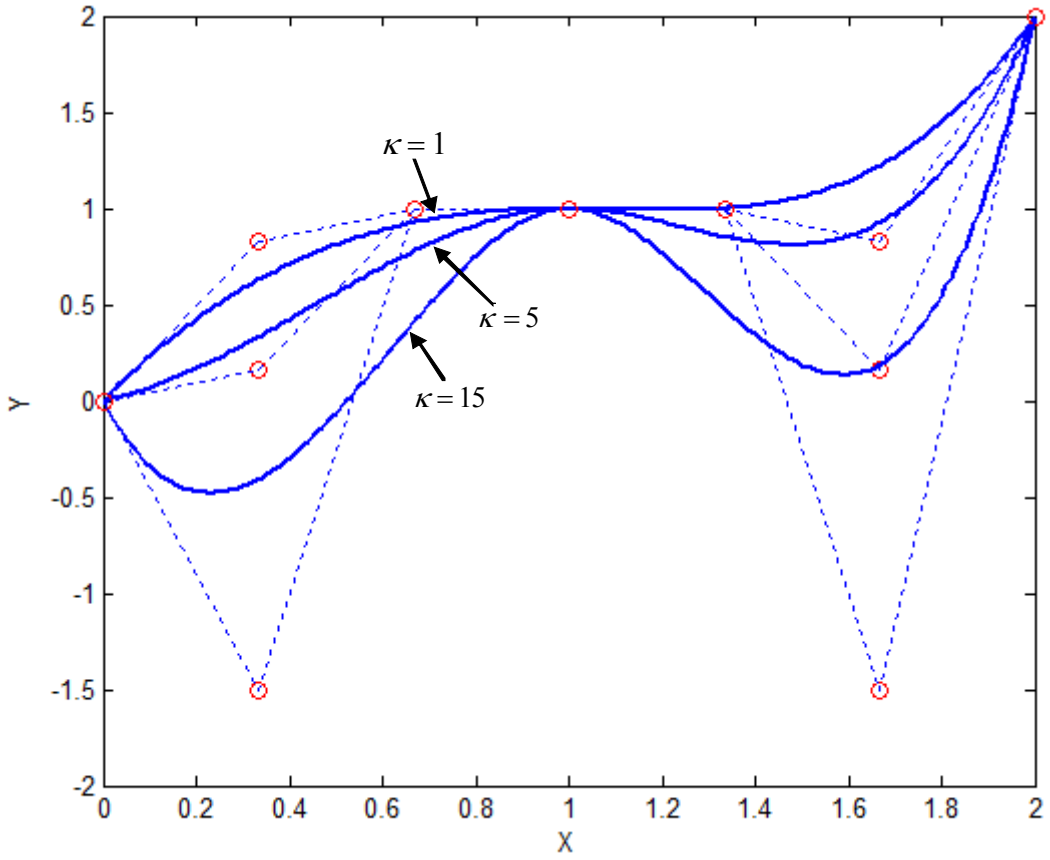


Figure 7.30. Bezier Curves with Curvature Specification

Similarly, higher-order parametric derivatives can be defined to satisfy higher-order geometric constraints. It is useful to look into these curve generation techniques as the resulting curves come with several positive properties (Convex Hull, Variation Diminishing, etc) that provide some determination of curve behavior. Likewise, some of the other techniques discussed in Chapter 3 (such as B-Splines, Beta Splines, and A-Splines) could be solved to meet specific parametric constraints. This allows for the same physical meanings defined as part of this work to be formulated into other existing curve generation schemes. A more thorough comparison of these techniques as well as Polynomial and Trapezoidal specifications could be an interesting area for future research.

7.3.2. Rotational Motion Specification

While Chapter 6 provided a brief overview of some different ways to describe rotational motions, a more in-depth look at this problem is needed. This section will provide several potential areas of future work. One potentially interesting area of work discussed in Section 6.3.3.2 was using the motion of the Frenet Frame along the curve to define the rotational motion. However, this method has some drawbacks. For one, it is difficult to predict how the frame will vary along a curve with large higher-order properties. Second, the frame can hit singularities and points where it is undefined (as in when curvature vanishes). One potential way to avoid this problem while still relating the rotational motion to the geometry of the curve is using parallel transport frames [11][19].

These frames take advantage of the fact that only the unit tangent vector is actually attached to the geometry of the curve. Then, a set of vectors $\left\{\hat{\mathbf{N}}_1, \hat{\mathbf{N}}_2\right\}$ perpendicular to the tangent can be defined to vary smoothly along the curve (Equation 7.22). This will lead to a non-singular rotational motion along the curve since the tangent vector should never vanish. A full algorithm for how to develop these curves is provided in [19] and analogous methods for quaternion frames are shown in [20][21].

$$\begin{bmatrix} \dfrac{d\hat{\mathbf{T}}}{ds} \\ \dfrac{d\hat{\mathbf{N}}_1}{ds} \\ \dfrac{d\hat{\mathbf{N}}_2}{ds} \end{bmatrix} = \begin{bmatrix} 0 & k_1 & k_2 \\ -k_1 & 0 & 0 \\ -k_2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{T}} \\ \hat{\mathbf{N}}_1 \\ \hat{\mathbf{N}}_2 \end{bmatrix} \qquad \textbf{7.22}$$

While in many applications (such as pick and place operations) it may be desirable to base rotational motion on the translational geometry, it is often better to plan rotational motions independent of the translation. Several methods for performing orientation-to-orientation interpolation were demonstrated in Chapter 6 as well as one

method for describing a motion through multiple quaternions. However, it may also be possible to develop a method for rotational planning analogous to the translational work done in this report.

This could be accomplished by defining local rotational motions at frames of interest. This would probably involve defining the instantaneous velocities in either the world frame $\{\dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z\}$ or the local frame (frenet or tool) $\{\dot{\theta}_T, \dot{\theta}_N, \dot{\theta}_B\}$. Another possibility is to define an axis and angular velocity (as in an Equivalent Axis formulation). This will allow local rotational motion to be described around a specific axis. This axis could once again be defined either in the world frame or some local frame. Then, these rotational positions and velocities can be blended together in a number of ways.

7.3.3. Software Implementation

As the software implementation is mainly meant as a testbed for this research, the main improvements that can be made would be to include any of the additional methods described in the last few sections. For example, the choice of low-level curve generation method (polynomial, trapezoidal, Bezier, etc) could be specified by the user. This would allow for an easier comparison between different methods. Also, the velocity profiles currently implemented are very simple and could be improved. However, with the implementation of more complex profiles, the approximation scheme for interpolating the geometric parameter *u* (Equation 7.15) may also need to be improved.

7.3.4. Future Applications

In the last section, a number of improvements that can be made to the existing framework were suggested. However, a more important area of future work is to apply these results to actual physical systems and tasks. This will hopefully allow for a better

understanding of the physical capabilities at both a task level and a manipulator level. The following sections will briefly introduce some possible application areas.

7.3.4.1. Task-Based Planning

One potential area of future work will be to relate the geometric properties of curves to physical task-based properties. For example, the relationship shown in Equation 7.23 was first developed in Section 2.3.2. This equation shows that the acceleration of a particle along a curve can be defined by its current frame (the tangent and normal vectors), its current speed/acceleration ($\dot{s}/\ddot{s}$) and its curvature κ. Thus, if the end-effector is moving at a constant speed, the magnitude of its centripetal acceleration would be $\kappa\dot{s}^2$. Thus, this magnitude can be plotted as function of curvature and speed as shown in Figure 7.31. While this relationship is quite simple, it shows that the geometric properties of curves can be related to actual physical phenomena.

$$\mathbf{a}(t) = \hat{\mathbf{T}}\ddot{s} + \kappa\hat{\mathbf{N}}\dot{s}^2 \qquad \textbf{7.23}$$

Another simple example of this is to use the Darboux vector as described in Section 6.3.3.2. The magnitude of this vector ($\sqrt{\kappa^2+\tau^2}$) gives the magnitude of the angular velocity of the Frenet Frame as it moves along a curve. Thus, if the rotational motion of the end-effector is based on the geometric path (e.g. as in surface polishing), the angular velocity can be directly calculated based on the geometric properties of curvature and torsion. A visual representation of this is shown in Figure 7.32.

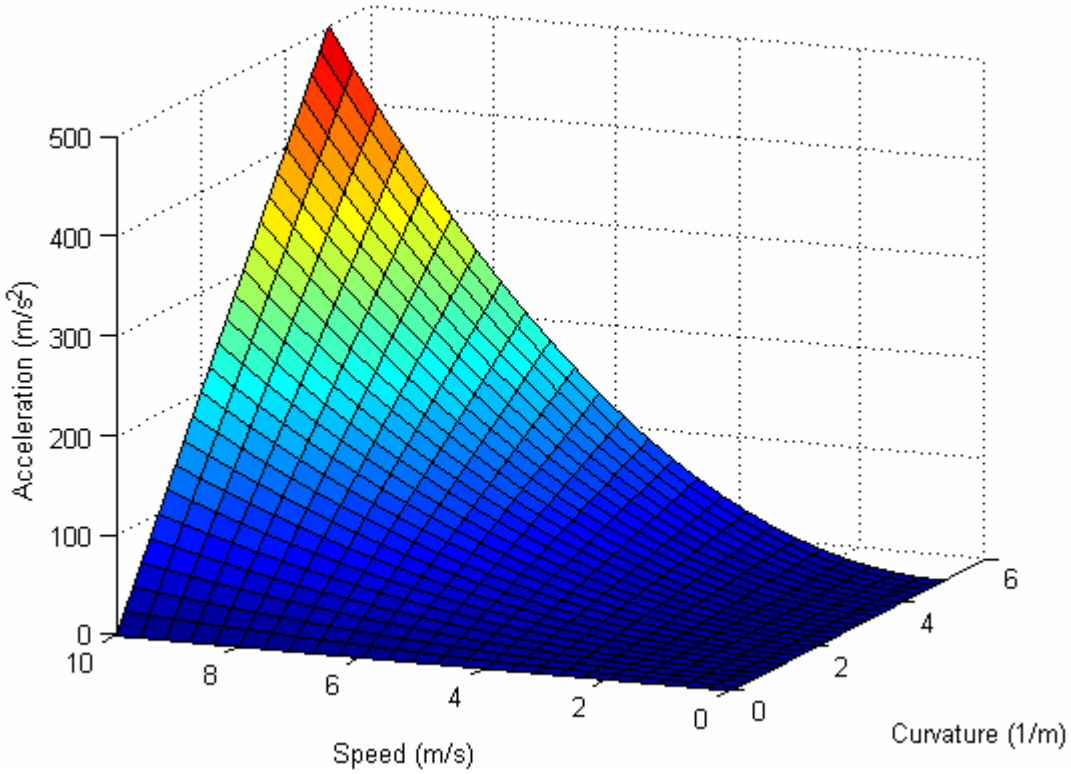


Figure 7.31. Centripetal Acceleration Plot

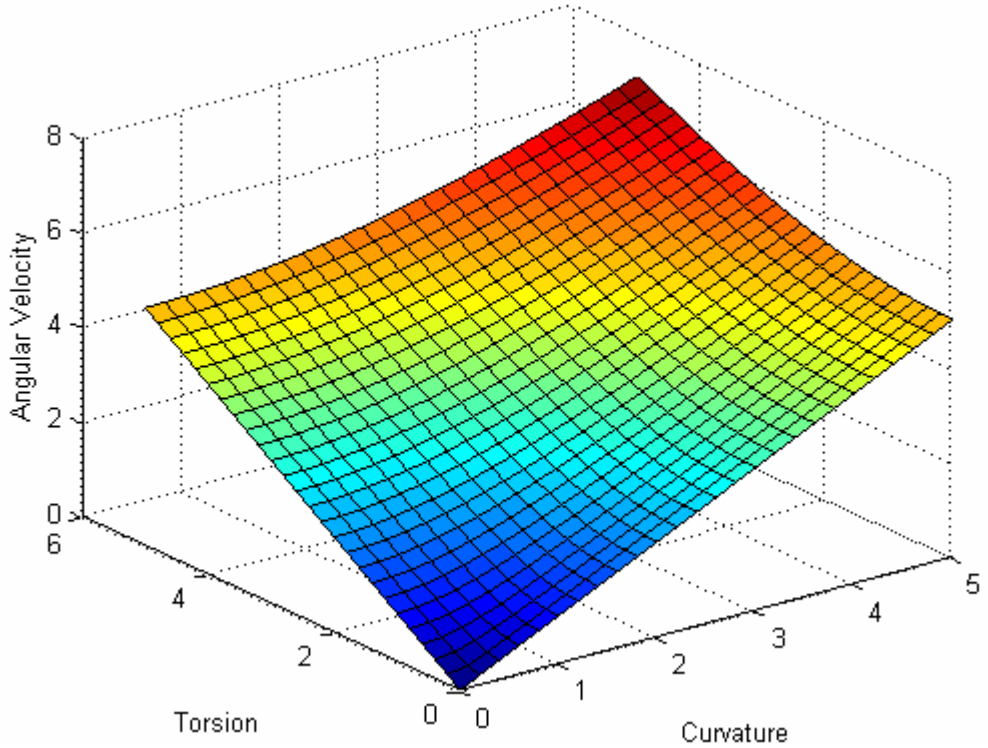


Figure 7.32. Angular Velocity Plot

The two examples above are very simple and intuitive. However, they could still be of use. For example, if a task requires a certain velocity and has a maximum acceleration at a particular point, a maximum allowable curvature can be defined. This then becomes a constraint on the geometric parameters. Thus, by studying the physical demands and properties required for a variety of tasks, constraints can be developed on the geometric parameters. For some tasks, specific values of these parameters will be required at particular points. In other tasks, a range of allowable values may be developed that gives the operator or user some degree of freedom in the curve specification.

7.3.4.2. G and H Parameters

In the last section, it was shown how the geometric properties of spatial curves can affect the physical task-level properties. Now, a further extension of this to the system-level properties will be presented. These relationships were first developed in an earlier work at the RRG [32]. This model consists of kinematic influence coefficients which are based only on the geometry of the system [51] (first presented in Section 1.1.3). The relationships between input and output velocities and accelerations that resulted from this work are shown in Equations 7.24 and 7.25.

$$\mathbf{v}_p = \left[G_p \right] \dot{\phi} \tag{7.24}$$

$$\mathbf{a}_p = \dot{\phi}^T \left[H_p \right] \dot{\phi} + \left[G_p \right] \ddot{\phi} \tag{7.25}$$

The first of these equations shows the relationship between the output velocities at the end-effector and the input velocities at the joints. These are related by the first order influence coefficients, where $\left[G_p \right]_n = \frac{\partial \mathbf{v}_p}{\partial \phi_n}$. The relationship between the accelerations is shown in Equation 7.25. For this, the second-order influence coefficients are also needed, where $\left[H_{jk} \right]_{m;n} = \frac{\partial}{\partial \phi_m} \left(\left[G_{jk} \right]_n \right)$.

By substituting these equations into the results the equations for motion along a spatial curve, the relationships between the input parameters of a serial manipulator and curve properties can be expressed. For example, using the relationship $\mathbf{v}_p = \frac{d\mathbf{p}}{ds}\frac{ds}{dt} = \frac{d\mathbf{p}}{ds}\dot{s}$ with Equation 7.24 leads to Equation 7.26.

$$\mathbf{v}_p = \left[G_p\right]\dot{\phi} = \frac{d\mathbf{p}}{ds}\dot{s} \qquad \textbf{7.26}$$

By moving the $\dot{s}$ to the left hand side of this equation, the unit tangent vector can be expressed in terms of the *G* functions and input joint velocities as shown in Equation 7.27.

$$\hat{\mathbf{T}} = \left[G_p\right]\frac{\dot{\phi}}{\dot{s}} \qquad \textbf{7.27}$$

Similarly, a representation for the Normal Vector, **N**, can be found (note that this is not the Unit Normal, $\hat{\mathbf{N}}$). This is shown in Equation 7.28. Also, , we know that the magnitude of Equation 7.28 is equal to the curvature, κ. Thus, the curvature can be completely defined by the input parameters of the system.

$$\mathbf{N} = \frac{d^2\mathbf{p}}{ds^2} = \frac{\frac{d^2\mathbf{p}}{dt^2} - \frac{d\mathbf{p}}{ds}\ddot{s}}{\dot{s}^2} = \frac{\dot{\phi}^T\left[H_p\right]\dot{\phi} + \left[G_p\right]\ddot{\phi} - \left[G_p\right]\frac{\dot{\phi}}{\dot{s}}\ddot{s}}{\dot{s}^2} \qquad \textbf{7.28}$$

Using Equations 7.27 and 7.28, the Bi-Normal vector can also be described in terms of input parameters as shown in Equation 7.29. This allows for the entire Frenet Frame as well as the curvature to be described in terms of input parameters.

$$\mathbf{B} = \mathbf{T}\times\mathbf{N} = \begin{pmatrix} 0 & -[G^z]\frac{\dot{\phi}}{\dot{s}} & [G^y]\frac{\dot{\phi}}{\dot{s}} \\ [G^z]\frac{\dot{\phi}}{\dot{s}} & 0 & -[G^x]\frac{\dot{\phi}}{\dot{s}} \\ -[G^y]\frac{\dot{\phi}}{\dot{s}} & [G^x]\frac{\dot{\phi}}{\dot{s}} & 0 \end{pmatrix} \begin{pmatrix} \frac{\dot{\phi}^T\left[H^x\right]\dot{\phi} + \left[G^x\right]\ddot{\phi} - \left[G^x\right]\frac{\dot{\phi}}{\dot{s}}\ddot{s}}{\dot{s}^2} \\ \frac{\dot{\phi}^T\left[H^y\right]\dot{\phi} + \left[G^y\right]\ddot{\phi} - \left[G^y\right]\frac{\dot{\phi}}{\dot{s}}\ddot{s}}{\dot{s}^2} \\ \frac{\dot{\phi}^T\left[H^z\right]\dot{\phi} + \left[G^z\right]\ddot{\phi} - \left[G^z\right]\frac{\dot{\phi}}{\dot{s}}\ddot{s}}{\dot{s}^2} \end{pmatrix} \qquad \textbf{7.29}$$

The fomulations provided in this section show a beginning of how to relate geometric properties to the actually physical system inputs. There is still a lot of possible work in this area. For one, the relationships developed above only take into account the translational (and not rotational) motion along a curve. For the cases where rotation is planned completely independent of translation, these relationships should be easy to define. However, for rotational motions that are tied to some specific geometry, this may become more difficult. Also, not all of the geometric properties studied in this research have been formulated in terms of system inputs. However, a more important first step of potential research in this area is to demonstrate that these relationships can be used in the planning phase to develop geometric paths that meet system capabilities.

7.4. CONCLUDING REMARKS

The research presented in this report provides a geometric framework for describing spatial curves based on constraints with physical meaning (curvature, torsion, and their derivatives). This differs from most current techniques in that these methods often involve interactively tweaking control points or parameters to create a visually pleasing shape. The benefit of the method presented in this report is that these physical constraints provide a better relationship to the physical motion of a manipulator. While only the geometric framework for describing these curves was developed in this work, it is hoped that future work built on this foundation will lead to a better way of defining manipulator motions that can take into account both the task and manipulator performance.

APPENDIX A

Calculation of A-Spline Coefficients

First, the scaffold for the section of the curve ($y^2 - x^3 = 0$) to be captured is chosen as $p_1 = [0,0]$, $p_2 = [1,1]$, and $p_3 = [1,0]$. Then, the relationship given in Equation 3.18 is inverted to determine $(\alpha_1, \alpha_2, \alpha_3) = f(x, y)$ as shown in Equation A.1.

$$\begin{bmatrix} p_1 & p_2 & p_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1-x \\ y \\ x-y \end{bmatrix} \quad \textbf{A.1}$$

Then, these values for α (in terms of *x* and *y*) can be plugged into Equation 3.19 to determine the basis fuctions. For this case, these basis functions are as shown in Equation A.2.

$$\begin{aligned}
B^3_{003}(x,y) &= x^3 - 3x^2y + 3xy^2 - y^3 \\
B^3_{012}(x,y) &= 3x^2y - 6xy^2 + 3y^3 \\
B^3_{021}(x,y) &= 3xy^2 - 3y^3 \\
B^3_{030}(x,y) &= y^3 \\
B^3_{102}(x,y) &= -3x^3 + 6x^2y - 3xy^2 + 3x^2 - 6xy + 3y^2 \\
B^3_{111}(x,y) &= -6x^2y + 6xy^2 + 6xy - 6y^2 \\
B^3_{120}(x,y) &= -3xy^2 + 3y^2 \\
B^3_{201}(x,y) &= 3x^3 - 3x^2y - 6x^2 + 6xy + 3x - 3y \\
B^3_{210}(x,y) &= 3x^2y - 6xy + 3y \\
B^3_{300}(x,y) &= -x^3 + 3x^2 - 3x + 1
\end{aligned} \quad \textbf{A.2}$$

Now, the coefficients of these polynomials can be extracted and represented as shown in Equation A.3

$$\begin{bmatrix} B^3_{003}(x,y) \\ B^3_{012}(x,y) \\ B^3_{021}(x,y) \\ B^3_{030}(x,y) \\ B^3_{102}(x,y) \\ B^3_{111}(x,y) \\ B^3_{120}(x,y) \\ B^3_{201}(x,y) \\ B^3_{210}(x,y) \\ B^3_{300}(x,y) \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & -3 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 6 & -3 & 3 & 3 & -6 & 0 & 0 & 0 \\ 0 & 0 & -6 & 6 & 0 & 0 & -6 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & -6 & 6 & 0 & 6 & -6 & 0 \\ 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & -6 & 0 & 3 & 0 \\ -1 & 0 & 0 & 0 & 3 & 0 & 0 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^3 \\ y^3 \\ x^2y \\ xy^2 \\ x^2 \\ y^2 \\ xy \\ x \\ y \\ 1 \end{bmatrix} \quad \textbf{A.3}$$

Now, the desired combination for the polynomial basis functions of the original implicit equation are determined. In this case, the y^2 term is 1 and the x^3 term is -1. Now, a system of linear equations can be solved to determine the b_{ijk} coefficients.

$$\begin{bmatrix} b_{003} \\ b_{012} \\ b_{021} \\ b_{030} \\ b_{102} \\ b_{111} \\ b_{120} \\ b_{201} \\ b_{210} \\ b_{300} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & -3 & 0 & 0 & 3 & 0 & -1 \\ -1 & 3 & -3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & 6 & -6 & 0 & -3 & 3 & 0 \\ 3 & -6 & 3 & 0 & -3 & 6 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & -6 & 0 & 3 \\ 0 & 0 & 0 & 0 & 3 & -6 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -6 & 6 & 0 & 6 & -6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{A.4}$$

Solving this equation yields the result from before that $b_{003} = -1$, $b_{012} = -1$, $b_{021} = -\frac{2}{3}$, $b_{120} = \frac{1}{3}$, and all other coefficients are zero. It should be noted that this system of equations can be used to solve any $d \le 0$ implicit curve laying in the same domain (same p_1, p_2, and p_3).

APPENDIX B

Evaluation of Trajectory Blending Techniques

B.1. Introduction

This appendix will examine more closely two methods for generating trajectories between the parametric constraints $(\frac{dx}{du},\frac{dy}{du},\frac{dz}{du},\cdots,\frac{d^n x}{du^n},\frac{d^n y}{du^n},\frac{d^n z}{du^n})$ developed in Chapter 5: polynomial and trapezoidal. First, the basic formulations of these methods will be described. Then, these techniques will be compared on their ability to control the high-order properties (parametric and geometric) through an example. While these methods do not represent every possible way of meeting the developed parametric constraints, they provide a good starting point for examining this problem.

B.2. Polynomial Trajectory Formulation

Polynomial trajectories are one of the most basic formulations to use as a parametric curve generation tool. A generalized form of a 1-DOF parametric polynomial is shown in Equation B.1 where *n* is the order of the polynomial. For example, an expanded third-order (cubic) polynomial is shown in B.2

$$p(u)=\sum_{i=0}^{n} a_i u^i \qquad \textbf{B.1}$$

$$p(u)=a_0+a_1 u+a_2 u^2+a_3 u^3 \qquad \textbf{B.2}$$

The coefficients $\{a_0,a_1,\cdots,a_n\}$ of the parametric polynomial can be solved by providing *n*+1 constraints for the function. For example, a cubic could be defined that meets four different function values $\{p_0,p_1,p_2,p_3\}$ at four different parameter values $\{u_0,u_1,u_2,u_3\}$. The coefficients for this can be solved using the system of linear equations shown in Equation B.3. All of the values in these equations are specified except for the

coefficients $\{a_0, a_1, a_2, a_3\}$. Thus, the square matrix can be inverted and moved to the other side of the equation to solve for the coefficients.

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 1 & u_0 & u_0^2 & u_0^3 \\ 1 & u_1 & u_1^2 & u_1^3 \\ 1 & u_2 & u_2^2 & u_2^3 \\ 1 & u_3 & u_3^2 & u_3^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad \textbf{B.3}$$

However, a more relevant way to use this method is shown in Equation B.4. In this system of equations, the four available constraints are used to satisfy values of the function as well as its first derivative at two parameter values.

$$\begin{bmatrix} p_0 \\ \frac{dp_0}{du} \\ p_1 \\ \frac{dp_1}{du} \end{bmatrix} = \begin{bmatrix} 1 & u_0 & u_0^2 & u_0^3 \\ 0 & 1 & 2u_0 & 3u_0^2 \\ 1 & u_1 & u_1^2 & u_1^3 \\ 0 & 1 & 2u_1 & 3u_1^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad \textbf{B.4}$$

A more generalized form of this equation is shown in Equation B.5. This shows that to meet constraints up to the n^{th} derivative at both ends of a trajectory will require a polynomial of order $2n+1$. In most applications, the trajectory is broken into smaller pieces to allow lower order polynomials to fit the constraints. However, in this research, constraints have been developed all the way to the 4th derivative. Thus, using a higher-order polynomial is unavoidable.

$$\begin{bmatrix} p_0 \\ \frac{dp_0}{du} \\ \vdots \\ \frac{d^n p_0}{du^n} \\ p_1 \\ \frac{dp_1}{du} \\ \vdots \\ \frac{d^n p_1}{du^n} \end{bmatrix} = \begin{bmatrix} 1 & u_0 & u_0^2 & u_0^3 & \cdots & u_0^{2n+1} \\ 0 & 1 & 2u_0 & 3u_0^2 & \cdots & (2n+1)u_0^{2n} \\ & & & \vdots & & \\ 1 & u_1 & u_1^2 & u_1^3 & \cdots & u_1^{2n+1} \\ 0 & 1 & 2u_1 & 3u_1^2 & \cdots & (2n+1)u_1^{2n} \\ & & & \vdots & & \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{2n+1} \end{bmatrix} \quad \textbf{B.5}$$

B.2. Trapezoidal Trajectory Formulation

Trapezoidal specification is another method for generating smooth 1-DOF trajectories. This method involves defining one of the derivatives of the parametric curve to have a trapezoidal shape using piece-wise singularity functions (Equation B.6). These singularity functions allow for the shape of the trapezoid to be defined in small pieces. For example, Figure B.1 shows a trapezoidal profile defined at the third derivative, where $\{u_0, u_1, u_2, \cdots, u_7\}$ are known as the breakpoints. The shape of the trajectory profile is defined differently in each segment between consecutive breakpoints. This trapezoidal shape can then be integrated to solve for the lower-order derivatives.

$$\langle u - u_i \rangle = \begin{cases} 0, & u < u_i \\ 1, & u \geq u_i \end{cases} \quad \textbf{B.6}$$

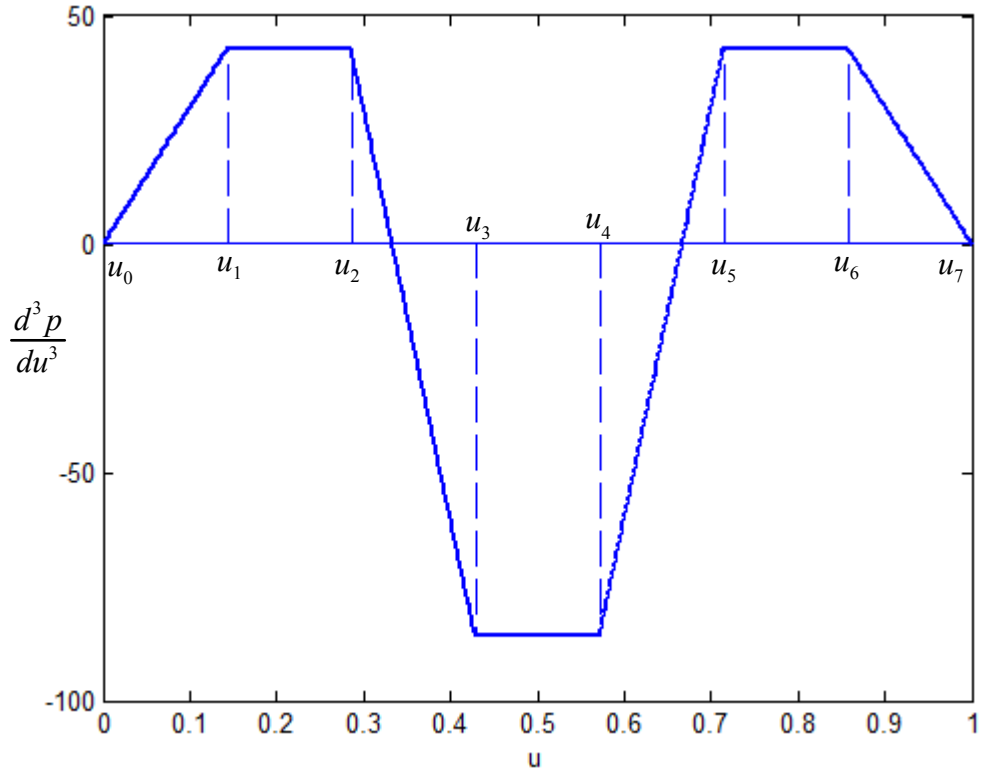


Figure B.1. Example Trapezoidal Specification

A generalized method for defining trapezoidal profiles was developed by Tesar and Matthews[50] for use in generating cam trajectories. Equation B.7 shows the generalized formulation of a trapezoidal method of generic order. The *n* parameter in this equation is the order of the profile (the derivative that the trapezoidal shape is defined in) and the *i* parameter defines the derivative to be evaluated. For example, for a third-order (*n*=3) system, setting *i*=3 would solve for the position function. The $\bar{\theta}_{ij}$ values are calculated using the breakpoints and singularity functions as shown in Equation B.8.

$$\frac{d^{n-i}p}{du^{n-i}}(u) = \sum_{k=1}^{i} \frac{\frac{d^{n-k}p_0}{du^{n-k}}u^{i-k}}{(i-k)!} + \sum_{j=1}^{n} \frac{A_j}{(i+1)!}\bar{\theta}_{ij} \qquad \textbf{B.7}$$

$$\theta_{ij} = \left[\frac{\langle u-u_{2j-2}\rangle^{i+1} - \langle u-u_{2j-1}\rangle^{i+1}}{u_{2j-1}-u_{2j-2}} - \frac{\langle u-u_{2j}\rangle^{i+1} - \langle u-u_{2j+1}\rangle^{i+1}}{u_{2j+1}-u_{2j}} \right] \qquad \textbf{B.8}$$

As mentioned before, the breakpoints $\{u_0, u_1, u_2, \cdots, u_{2n+1}\}$ can be defined by the user and provide some control over the shape of the profile. The coefficients A_j are calculated based on the initial and final conditions as shown in Equation B.9. A more detailed derivation of these equations can be found in Tesar and Matthews[50].

$$\bar{\theta}_{ij}\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} = \begin{bmatrix} 2\dfrac{d^{n-1}p_1}{du^{n-1}} \\ \vdots \\ (i+1)!\dfrac{d^{n-i}p_1}{du^{n-i}} \end{bmatrix} - \begin{bmatrix} 2\dfrac{d^{n-1}p_0}{du^{n-1}} \\ \vdots \\ (i+1)!\displaystyle\sum_{k=1}^{i}\dfrac{\dfrac{d^{n-i}p_0}{du^{n-i}}u_n^{i-k}}{(i-k)!} \end{bmatrix} \qquad \textbf{B.9}$$

B.3. Example Trajectory

Now, these trajectory generation methods can be applied to an example trajectory specification and the higher-order parametric and geometric properties along the curve can be examined. As in Chapter 5, first the geometric constraints $(\kappa, \kappa', \tau, \tau')$ for the motion are specified. Then, these constraints are formulated into parametric constraints $(\frac{dx}{du}, \frac{dy}{du}, \frac{dz}{du}, \cdots, \frac{d^n x}{du^n}, \frac{d^n y}{du^n}, \frac{d^n z}{du^n})$. Finally, both polynomial and trapezoidal methods will be used to develop individual *x*, *y*, and *z* trajectories that satisfy these constraints. A sample set of geometric constraints for this example is shown in Table B.1. This is a simple trajectory plan that specifies a value for curvature and torsion at the middle point.

	x	y	z	$\hat{\mathbf{T}}$	$\hat{\mathbf{N}}$	$\hat{\mathbf{B}}$	κ	$\frac{d\kappa}{du}$	τ	$\frac{d\tau}{du}$
$\mathbf{p}_1$	0.0	0.0	0.0	[1,0,0]	[0, 1,0]	[0,0, 1]	0.0	0.0	0.0	0.0
$\mathbf{p}_2$	2.0	1.0	1.0	$\begin{bmatrix} 0.0 \\ 0.707 \\ -0.707 \end{bmatrix}$	[-1,0,0]	$\begin{bmatrix} 0.0 \\ 0.707 \\ 0.707 \end{bmatrix}$	5.0	0.0	1.0	0.0
$\mathbf{p}_3$	1.0	3.0	-1.0	[0,1,0]	[-1,0,0]	[0,0,1]	0.0	0.0	0.0	0.0

Table B.1. Example Geometric Constraints

Table B.2 shows the calculated parametric constraints. The 4th order constraints for this specification are all zero because $\tau' = 0$. This somewhat simplifies the end constraints; however, the emphasis of this analysis is on the trajectory between the end points.

i	1	2	3	i	1	2	3	i	1	2	3
x_i	0.0	2.0	1.0	y_i	0.0	1.0	3.0	z_i	1.0	3.0	-1.0
$\frac{dx_i}{du}$	1.0	0.0	0.0	$\frac{dy_i}{du}$	0.0	0.707	1.0	$\frac{dz_i}{du}$	0.0	-0.707	0.0
$\frac{d^2x_i}{du^2}$	0.0	-4.9985	0.0	$\frac{d^2y_i}{du^2}$	0.0	0.0	0.0	$\frac{d^2z_i}{du^2}$	0.0	0.0	0.0
$\frac{d^3x_i}{du^3}$	0.0	0.0	0.0	$\frac{d^3y_i}{du^3}$	0.0	3.5339	0.0	$\frac{d^3z_i}{du^3}$	0.0	3.5339	0.0
$\frac{d^4x_i}{du^4}$	0.0	0.0	0.0	$\frac{d^4y_i}{du^4}$	0.0	0.0	0.0	$\frac{d^4z_i}{du^4}$	0.0	0.0	0.0

Table B.2. Calculated Parametric Constraints

Now, the individual trajectories can be developed. As mentioned earlier, the trapezoidal specification allows for some control over the shape of the trajectory through the breakpoints. For the first iteration of this example, these breakpoints were simply chosen to be spread between 0 and 1 uniformly as shown in Equation B.10[16]. Figure B.2 shows the higher-order parametric plots along the trajectory, and Figure B.3 and Figure B.4 show the curvature and torsion values, respectively. The plots of the parametric derivatives show that the two methods follow each other very closely with the trapezoidal profile having slightly higher peak values. The plots of curvature and torsion are also very similar with the polynomial trajectory having slightly higher peaks in the curvature plot, and the trapezoidal profile having slightly higher peaks in the torsion profile.

$$u_i = \{0,\ 0.111,\ 0.222,\ 0.333,\ 0.444,\ 0.556,\ 0.667,\ 0.778,\ 0.889,\ 1.00\} \qquad \textbf{B.10}$$

[16] This same configuration is used for the second half of the trajectory ($u \in [1, 2]$).

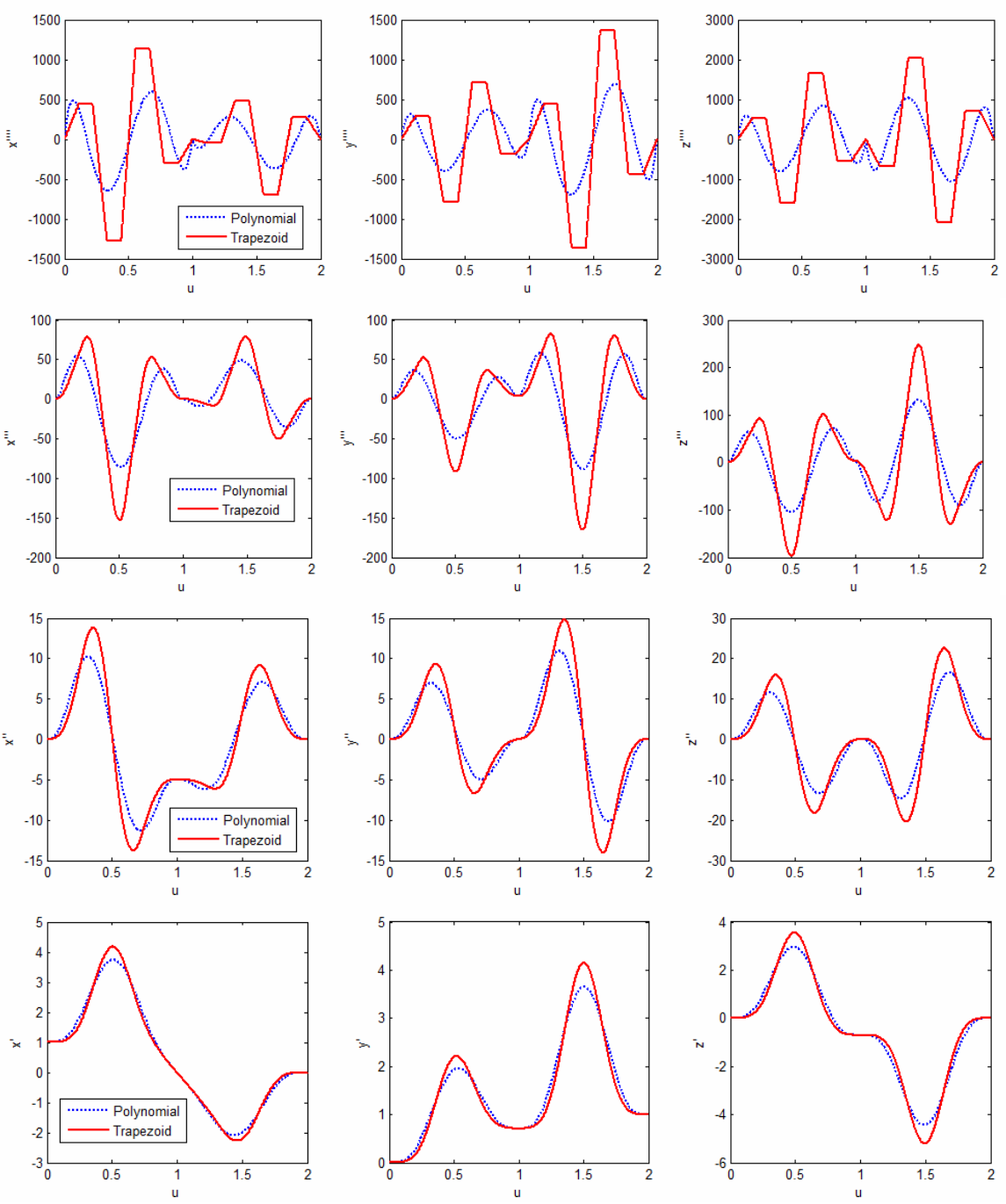


Figure B.2. Polynomial vs. Trapezoidal Specification Example I

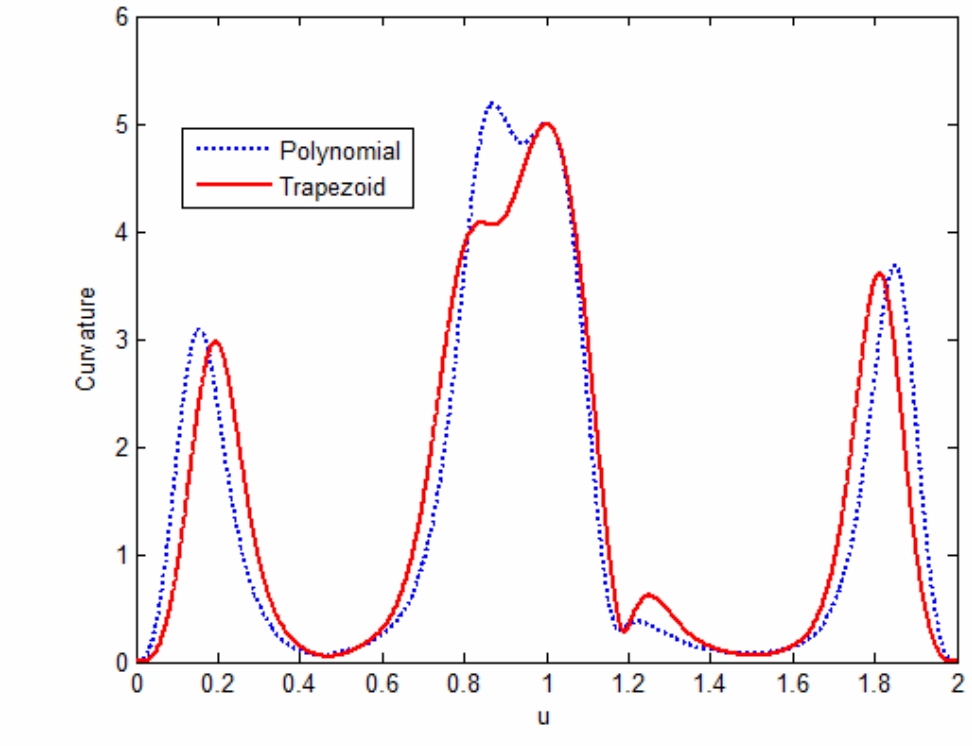


Figure B.3. Curvature Profiles I

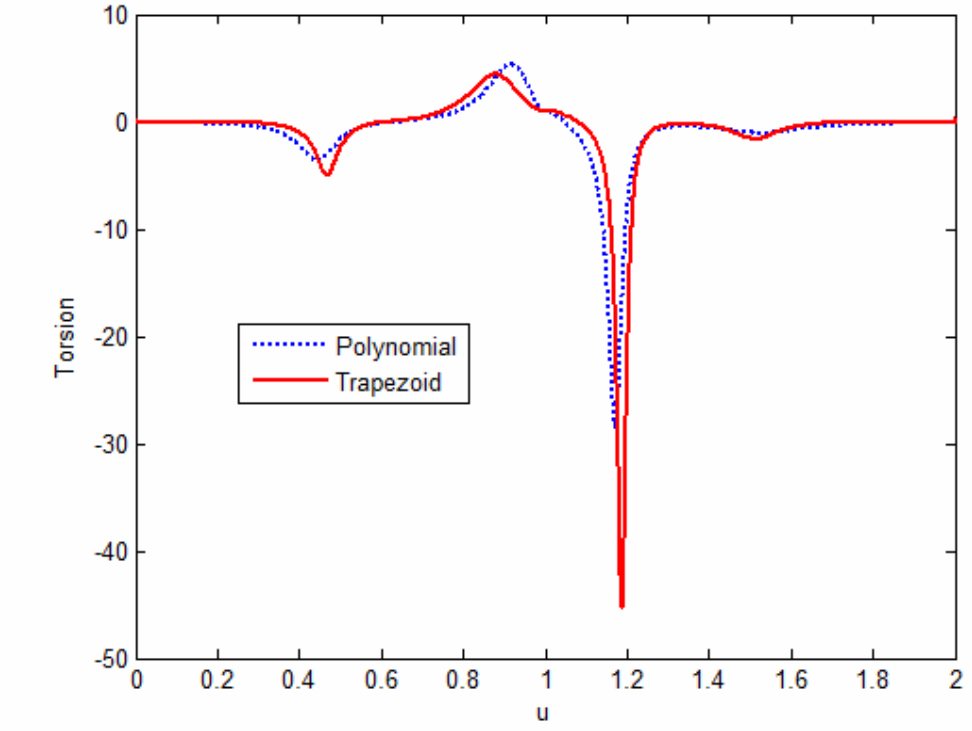


Figure B.4. Torsion Profiles I

Next, the breakpoints for the trapezoidal motion were manually modified in an attempt to lower the peak values of the parametric derivatives. This was done by moving the breakpoints that determine the ramp-up and ramp-down portions of the trapezoid closer together (as shown in B.11). This will decrease the magnitude of the peak value because it will increase the amount of time that it will stay at this value. Figure B.5 shows the resulting plots of the parametric derivatives. These plots show that the peak values of the trapezoidal profile are much closer to the polynomial profile though they are still slightly higher. Given a few more iterations of optimizing the breakpoint positions, these peaks could probably be brought even lower. The resulting curvature and torsion profiles are shown in Figure B.6 and Figure B.7. These plots show that the curvature results remain approximately the same while the peak value of the torsion plot of the trapezoidal profile decreased considerably from the last example. Thus, optimizing the breakpoint placement to decrease the peak values of the parametric derivatives appears to have some affect on the geometric properties (as expected). However, the high level coupling between the x, y, and z coordinates still makes this process difficult to predict.

$$u_i = \{0,\ 0.02,\ 0.24,\ 0.26,\ 0.49,\ 0.51,\ 0.74,\ 0.76,\ 0.98,\ 1.00\} \qquad \textbf{B.11}$$

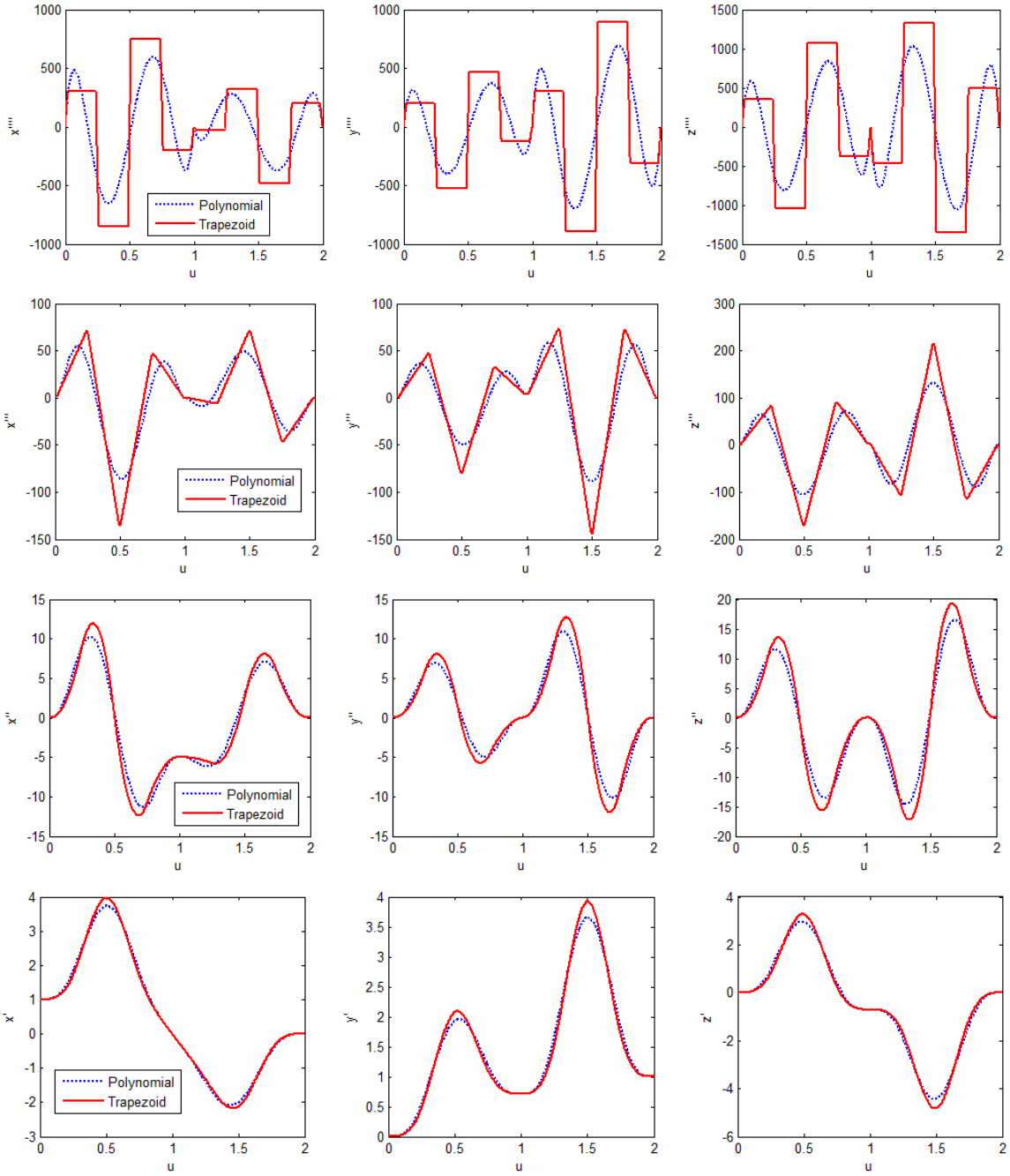


Figure B.5. Polynomial vs. Trapezoidal Specification Example II

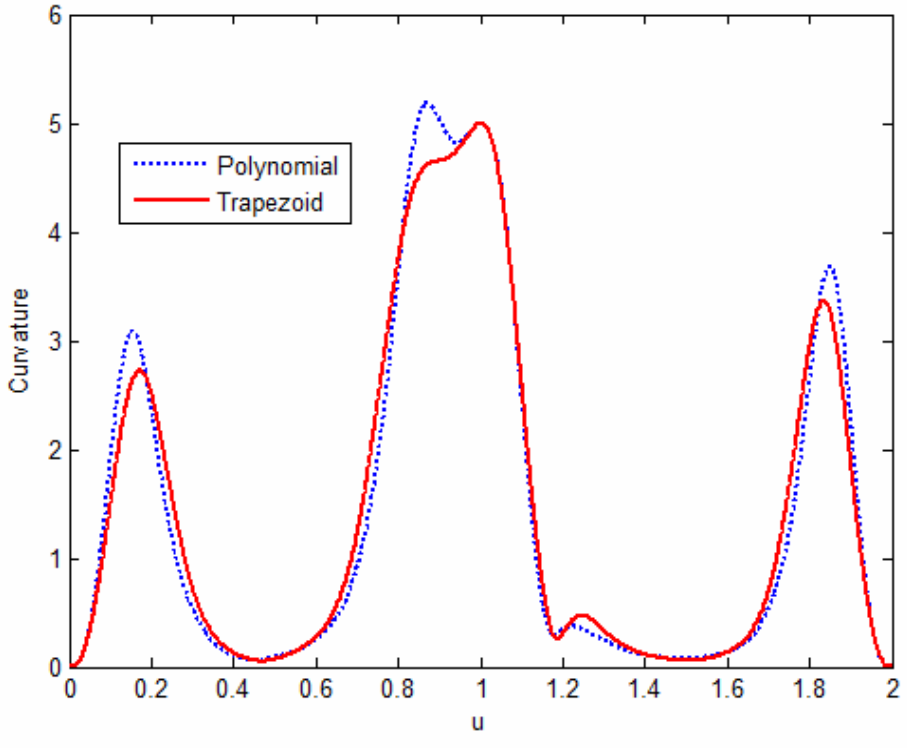


Figure B.6. Curvature Profiles II

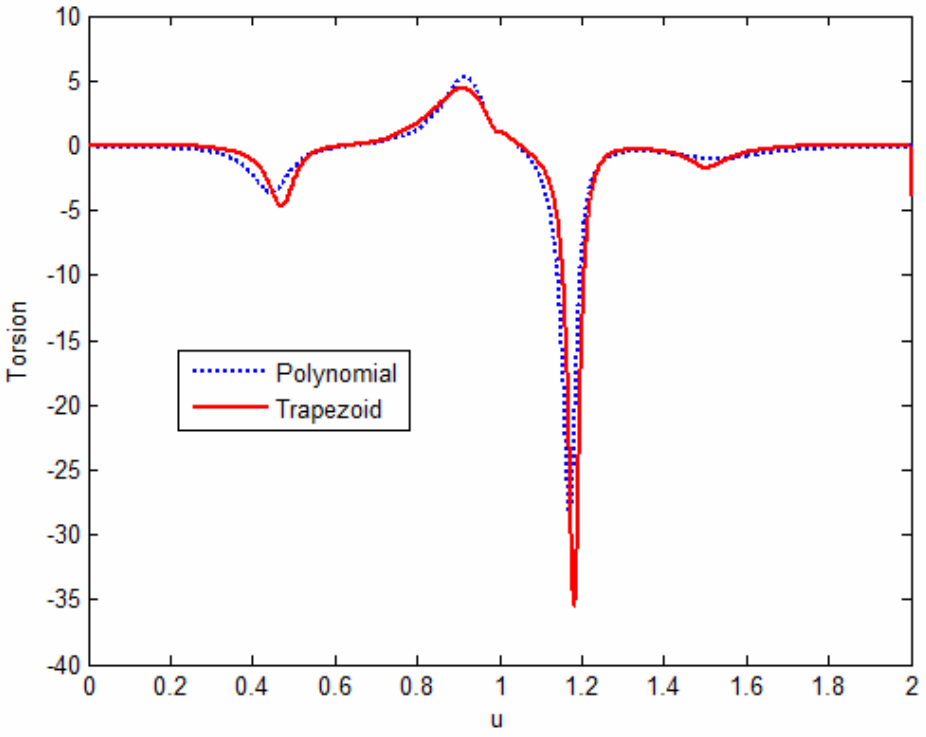


Figure B.7. Torsion Profiles II

As seen in the last two examples, the trapezoidal profile contained a higher peak value in the torsion profile. Thus, the last step of this iteration process is an attempt to manually modify the positions of the breakpoints to reduce this peak value. Equation B.12 shows the parametric equation for torsion. This shows that for any parameter value *u* the torsion is a function of the first, second, and third derivatives. Thus, the torsion may be reduced by altering where the peak values of these various derivatives are for each individual coordinate (x, y, z).

$$\tau(u) = \frac{\left(\frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du}\right) \cdot \frac{d^3\mathbf{p}}{du^3}}{\left\| \frac{d\mathbf{p}}{du} \times \frac{d^2\mathbf{p}}{du^2} \right\|^2} \qquad \textbf{B.12}$$

This was attempted using different breakpoints for each individual coordinate as shown in Equation B.13. This equation shows that the breakpoints of the x, y, and z directions are staggered at the beginning of the $u \in [1,2]$ trajectory (where the torsion peak occurs. This will keep the peak values of the parametric derivatives of each coordinate staggered as well and result in a lower value of torsion. Figure B.8 shows the parametric derivatives, and Figure B.9 and Figure B.10 show the curvature and torsion plots. The plot of torsion shows that this technique did lower the peak value of torsion at the desired point in the trajectory. However, it also created an extra smaller peak towards the end of the trajectory.

$$\begin{aligned} \mathrm{u}_\mathrm{i}^\mathrm{x} &= \{1.0,\ 1.02,\ 1.34,\ 1.36,\ 1.69,\ 1.71,\ 1.84,\ 1.86,\ 1.98,\ 2.00\} \\ \mathrm{u}_\mathrm{i}^\mathrm{y} &= \{1.0,\ 1.02,\ 1.24,\ 1.26,\ 1.49,\ 1.51,\ 1.74,\ 1.86,\ 1.98,\ 2.00\} \\ \mathrm{u}_\mathrm{i}^\mathrm{z} &= \{1.0,\ 1.02,\ 1.14,\ 1.16,\ 1.39,\ 1.41,\ 1.74,\ 1.86,\ 1.98,\ 2.00\} \end{aligned} \qquad \textbf{B.13}$$

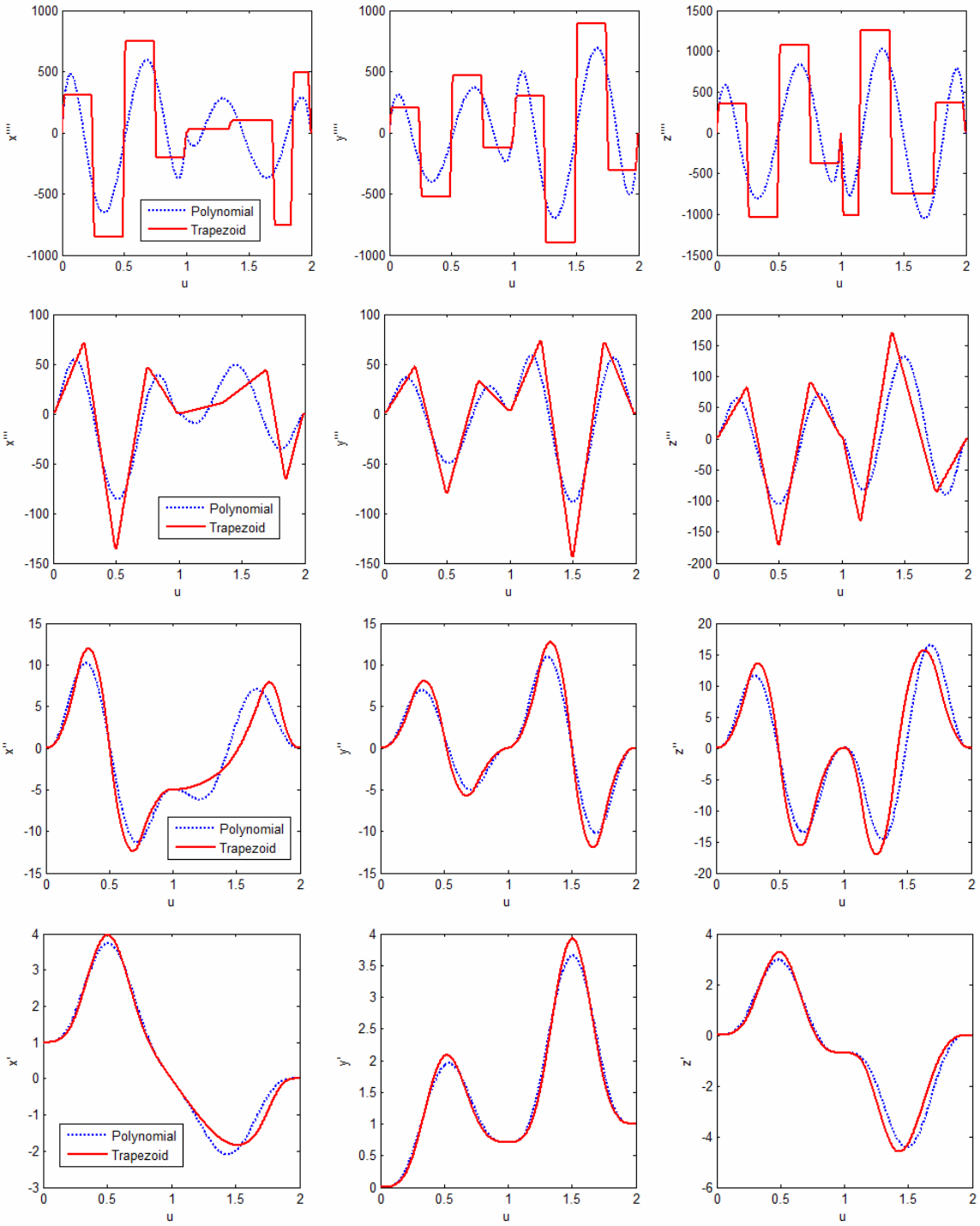


Figure B.8. Polynomial vs. Trapezoidal Specification Example III

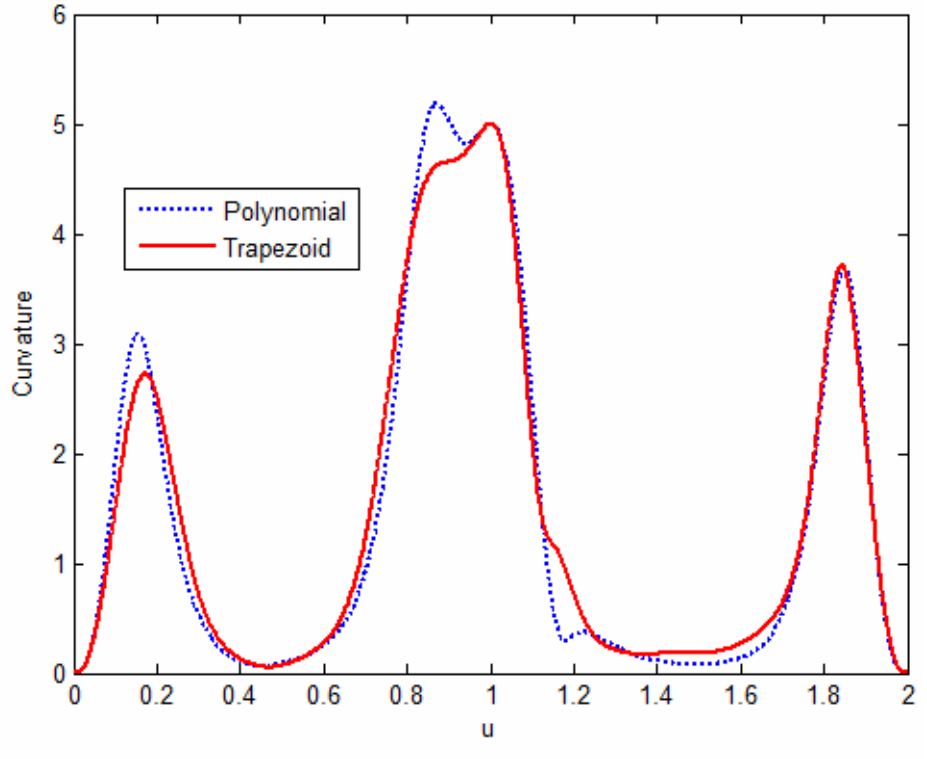


Figure B.9. Curvature Profiles III

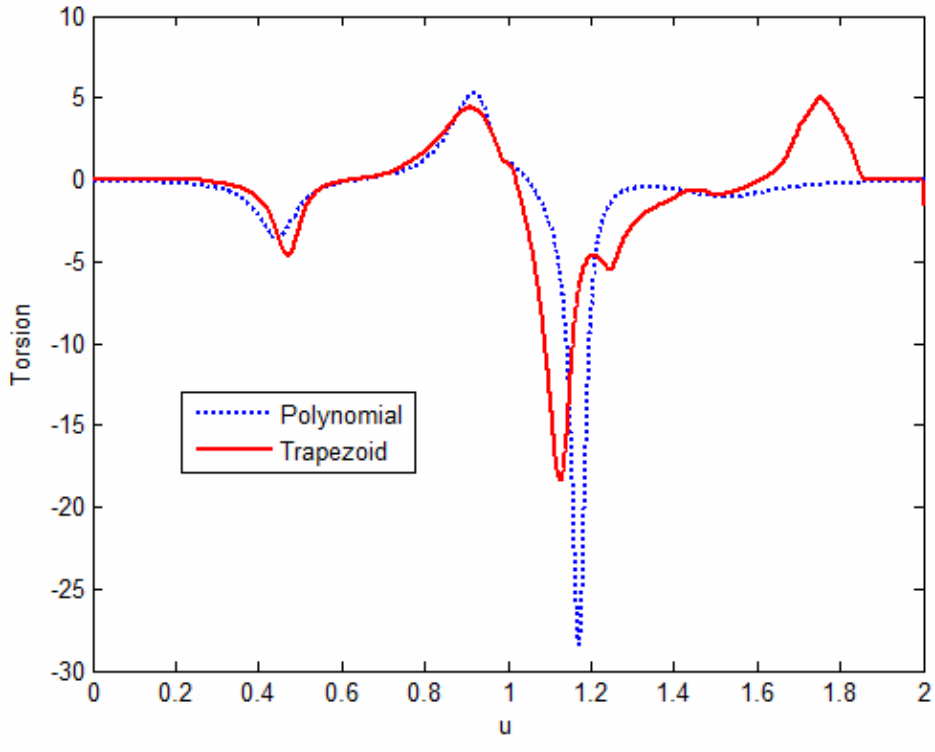


Figure B.10. Torsion Profiles III

B.4. Summary

This appendix examined two different methods of generating trajectories to satisfy parametric constraints: polynomial and trapezoidal. In the nominal case, these methods appear to be very similar. However, the ability to specify and modify the breakpoints in trapezoidal profiles allows for more control over the interior shape of the trajectory. This added control was demonstrated on a specific example to show how it could be used to lower the peak value of torsion. However, this method requires a high level of understanding of the underlying mathematics as well as a good amount of tweaking/iterating. This, as it is, this method is probably not suitable to be used for trajectory modification. However, techniques to optimize or automate this kind of process could be a useful area for future research.

APPENDIX C

Motion Planner Class Documentation

OSCAR::MotionPlanner Class Reference

Current Functionality: **MotionPlanner** is a class for performing Kinematics and generating trajectories for generic manipulators. The manipulator is defined through its DH Paramters, offsets, and limits. This class will then generate trajectories in either Joint Space or Cartesian space that satisfy the provided velocity and acceleration constraints.

Author:

Peter S. March

Public Methods

- **MotionPlanner** (const DHData &robotData, const Vector &jointOffsets, JointVector &initialJoints, Matrix &jointLimits, Vector &**velLimits**, Vector &**accLimits**, Vector &**handVelLimits**, Vector &**handAccLimits**, OSCARError &err=DUMMY_ERROR(noError))
- **~MotionPlanner** ()
- bool **PlanMove** (const JointVector **currentJoints**, const JointVector **targetJoints**, **MPTrajectoryType** trajType, double &moveTime)
- bool **PlanMove** (const JointVector **currentJoints**, const Xform targetHand, **MPTrajectoryType** trajType, double &moveTime)
- bool **PlanMoveJogJoint** (const JointVector &**currentJoints**, const JointVector &**currentVelocity**, const std::vector< double > directions)
- bool **PlanMoveJogCartesian** (const JointVector &**currentJoints**, const JointVector &**currentVelocity**, std::vector< double > directions)
- bool **PlanMoveVia** (const JointVector _currentJoints, std::vector< Vector > viaPoints)
- bool **PlanMoveVia** (const JointVector _currentJoints, std::vector< Xform > viaPoints, double moveTime, **MPViaType** viaType=FlyThrough)
- bool **PlanMoveViaGeometric** (const JointVector _currentJoints, std::vector< CurveParameter > viaPoints, double moveTime)
- bool **Stop** (const JointVector &**currentJoints**, const JointVector &**currentVelocity**, bool fastest=true)
- bool **GetJointPosition** (JointVector &jointPosVector, JointVector &**currentVelocity**, TrajectoryGenerator::TrajectoryState &state)
- bool **GetHandPosition** (JointVector &joints, Xform &handPosition)
- bool **GetHandPosition** (Xform &handPosition)
- bool **SetJointPosition** (const JointVector &joints)
- bool **SetToolPose** (const Xform &toolPose)
- bool **SetBasePose** (const Xform &basePose)
- bool **SetCycleRate** (double rate)
- bool **SetTrajectoryShape** (TrajectoryShapeType shape)
- bool **SetRampTime** (double **rampTime**)

- bool **SetCartesianControlMode** (CartesianCoordinateMode _coordMode)
- bool **SetSpeedScale** (double scale)
- bool **SetVelocityScale** (double scale)
- bool **SetAccelerationScale** (double scale)
- bool **ComputeHandVelocity** (const JointVector &**currentVelocity**, HandPose &handVel)

Protected Methods

- bool **checkLimits** (const JointVector &joints, bool checkVelocity=true)
- double **estimateMoveTime** (const Vector &start, const Vector &end)
- bool **getJointPosition** (std::vector< Vector > &jointPosBuffer, TrajectoryGenerator::TrajectoryState &state)
- bool **cancelMotion** ()

Protected Attributes

- IDNewtonEuler * **idnPtr**
- IDSANewtonEuler * **idsaPtr**
- FKJacobian * **fkjPtr**
- JointVector **currentJoints**
- JointVector **prevJoints**
- JointVector **prevJoints2**
- JointVector **currentVelocity**
- JointVector **prevVelocity**
- JointVector **currentAcceleration**
- JointVector **targetJoints**
- JointVector **tempJointPosVector**
- JointVector **tempJointPosVector2**
- std::list< Vector > **finalBuffer**
- std::string **robotName**
- TrajectoryGenerator::TrajectoryState **prevState**
- GeneralKinematicsHandler * **kinPtr**
- PathPlanner * **ppPtr**
- unsigned int **DOF**
- Vector **velLimits**
- Vector **accLimits**
- Vector **handVelLimits**
- Vector **handAccLimits**
- Vector **minVelLimits**
- Vector **minAccLimits**
- TrajectoryShapeType **trajShape**
- CartesianCoordinateMode **coordMode**
- double **cycleRate**
- double **velScale**
- double **accScale**
- double **rampTime**

- double **maxLinVel**
- double **timeScale**

Constructor & Destructor Documentation

OSCAR::MotionPlanner::MotionPlanner (const DHData & *robotData*, const Vector & *jointOffsets*, JointVector & *initialJoints*, Matrix & *jointLimits*, Vector & *velLimits*, Vector & *accLimits*, Vector & *handVelLimits*, Vector & *handAccLimits*, OSCARError & *err* = DUMMY_ERROR(noError))

Parameters:

dhData A DHData object that defines the DH parameters for the robot. The units of angles in the DHData should be Degrees. The robot DOF is determined from dhData.
offset A vector whose size should match the DOF defined in the dhData parameter. The values of offset should define the offset between the zero position of the robot as defined by the DH parameters and as represented by the physical robot zero position. Offset should be defined in Degrees for all joints that are revolute. If the DH paramater defined 0 position is the same as the real robot zero position, then all offsets will be zero.
jointLimits A Limits object that defines the position travel limits of the robot arm. The size of the limits object should be the same as the DOF defined in the dhData parameter. The limits should also be expressed in the robot coordinates instead of the DH parameter coordinates. For all revolute joints, the limits should be expressed in Degrees.
initialJoints The initial joint state of the manipulator. These joints will be used to initialize the Kinematics of the manipulator and must be a valid joint position. For all revolute joints, the initial position should be express in Radians.
velLimits The absolute value of the maximum joint velocities for each joint. These velocities should be express in Radians/s.
accLimits The absolute value of the maximum joint accelerations for each joint. These accelerations should be express in Radians/s^2.
handVelLimits The absolute value of the maximum cartesian velocities. The translational elements should be in mm/s and the rotational in Radians/s. These limits are used for teleoperation only.
handAccLimits The absolute value of the maximum cartesian accelerations. The translational elements should be in mm/s^2 and the rotational in Radians/s^2. These limits are used for teleoperation only.
err An OSCARError object that on return will hold the value of any errors that were generated during the constructor call. If err is not equal to 'noError' you can call GetError() to get the details of the error code.

Exceptions:

argumentSizeIncorrect #argumentSizeIncorrect. This error is generated when the size of the input parameter offset and limits does not match the DOF defined by the dhData parameter.

OSCAR::MotionPlanner::~MotionPlanner ()

Member Function Documentation

bool OSCAR::MotionPlanner::GetHandPosition (Xform & handPosition)

Use this to retrieve the internal Cartesian handpose state. This method will return the internal Cartesian state of the **MotionPlanner**. No calculations are performed.

Parameters:

handPosition The resulting hand position.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::GetHandPosition (JointVector & *joints*, Xform & *handPosition*)

Use this to calculate the hand position for a given joint configuration. This method can be used to perform the forward kinematics to retrieve a hand position for a provided joint configuration. This method does not change the internal state of the **MotionPlanner**.

Parameters:

joints The desired joint configuration.
handPosition The resulting hand position.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::GetJointPosition (JointVector & *jointPosVector*, JointVector & *currentVelocity*, TrajectoryGenerator::TrajectoryState & *state*)

Use this method to retrieve the current joint position/velocity. This method is used to continually get the current joint position and velocity state. By design, this method should be called repeatedly inside a loop running at the sample rate designated in **SetCycleRate()**. All computed velocities and accelerations are assuming the joint positions are being updated at this rate.

Parameters:

jointPosVector The current joint state of the manipulator in Radians.
currentVelocity The current velocity state of the manipulator in Rad/s.
state This returns the current state of the **MotionPlanner**. The valid values are:
Inactive - The manipulator is currently idle. In this state, the **MotionPlanner** will be just returning the current position over and over.
Active - The manipulator is current moving either through a point-to-point trajectory or through teleoperation.
TrajectoryComplete - The manipulator has just complete a point-to-point trajectory

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMove (const JointVector *currentJoints*, const Xform *targetHand*, MPTrajectoryType *trajType*, double & *moveTime*)

Generates a Trajectory to a target EEF position. This method will generate a trajectory to a target Joint Position. This trajectory can be either JointInterpolated or CartesianInterpolated.

Parameters:

currentJoints This is the current position of the manipulator in Radians.
targetHand An Xform containing the target hand position for the manipulator.
trajType This is set to either JointInterpolated or CartesianInterpolated. In the case of JointInterpolated, the final joint position is calculated from the final hand position
moveTime This parameter contains the move time for the trajector. If this move time is positive, a JointInterpolated move will return true/false based on if the trajectory can be completed without violating constraints while CartesianInterpolated move will will simply generate a trajectory for the given time (in this case, constraint errors will be found during execution). If the move time is 0.0, a JointInterpolated move will complete the move using the max velocity/acceleration constraints while a CartesianInterpolated move will estimate a fastest move time.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMove (const JointVector *currentJoints*, const JointVector *targetJoints*, MPTrajectoryType *trajType*, double & *moveTime*)

Generates a Trajectory to a target Joint Position This method will generate a trajectory to a target Joint Position. This trajectory can be either JointInterpolated or CartesianInterpolated.

Parameters:

currentJoints This is the current position of the manipulator in Radians.
targetJoints The target joint position for the manipulator.
trajType This is set to either JointInterpolated or CartesianInterpolated. In the case of CartesianInterpolated, the final hand position is calculated from the final joint position
moveTime This parameter contains the move time for the trajector. If this move time is positive, a JointInterpolated move will return true/false based on if the trajectory can be completed without violating constraints while CartesianInterpolated move will will simply generate a trajectory for the given time (in this case, constraint errors will be found during execution). If the move time is 0.0, a JointInterpolated move will complete the move using the max velocity/acceleration constraints while a CartesianInterpolated move will estimate a fastest move time.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMoveJogCartesian (const JointVector & *currentJoints*, const JointVector & *currentVelocity*, std::vector< double > *directions*)

Use this method to perform a Cartesian jog. This method can be used to jog the EEF as in teleoperation. When an axis is set to jog, it accelerates to its maximum velocity and coasts until another **PlanMoveJogCartesian()** or a **Stop()** is called.

Parameters:

currentJoints The current joint state of the manipulator in Radians.
currentVelocity The current velocity state of the manipulator in Rad/s.
directions A vector of length 6. The 6 values indiciate the x,y,z axes and the three angles in a FixedXYZ orientation description. The valid values are -1 to 1. If the value is

negative, the axis will decelerate from its current velocity to a of its maximum negative velocity. For example, a value of -0.5 will decelerate to 50% of the maximum velocity in the negative direction. If the value is positive, the axis will accelerate from its current velocity to a of its maximum positive velocity. For example, a value of 0.5 will accelerate to 50% of the maximum velocity in the positive direction. If the value is 0, the axis will decelerate to a zero velocity.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMoveJogJoint (const JointVector & *currentJoints*, const JointVector & *currentVelocity*, const std::vector< double > *directions*)

Use this method to jog the joints. This method can be used to jog the joints as in teleoperation. When a joint is set to jog, it accelerates to its maximum velocity and coasts until another **PlanMoveJogJoint()** or a **Stop()** is called.

Parameters:

currentJoints The current joint state of the manipulator in Radians.
currentVelocity The current velocity state of the manipulator in Rad/s.
directions A vector of length DOF. The valid values are -1 to 1 with each value corresponding to one joint. If the value is negative, the axis will decelerate from its current velocity to a of its maximum negative velocity as determined by the velocity limits and velocity scale. For example, a value of -0.5 will decelerate to 50% of the maximum velocity in the negative direction. Similarly, a positive value will accelerate to a of its maximum positive velocity (once again determined by hardware limits and velocity scale). If the value is 0, the axis will decelerate to a zero velocity.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMoveVia (const JointVector _*currentJoints*, std::vector< Xform > *viaPoints*, double *moveTime*, MPViaType *viaType* = FlyThrough)

Generates a Via Trajectory through multiple End-Effector positions. This method will generate a trajectory through a number of target End-Effector positions.

Parameters:

currentJoints This is the current position of the manipulator in Radians.
viaPoints A vector of Xform Via Positions.
moveTime The move time to complete the motion.
viaType Determines the method of interpolation through the via points. Options are FlyBy or FlyThrough. In FlyBy mode, straight line trajectories will be calculated between each via point, and then the transitions between two straight lines will be blended (i.e. "cutting the corner"). In FlyThrough mode, the generated trajectory will pass through each via point, but will also generate some overshoots. Default value is FlyThrough.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMoveVia (const JointVector _*currentJoints*, std::vector< Vector > *viaPoints*)

Generates a Via Trajectory through multiple joint positions. This method will generate a trajectory through a number of target joint positions.

Parameters:

currentJoints This is the current position of the manipulator in Radians.
viaPoints A vector of Joint Via Positions (in Radians). The generated path will pass through these positions.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::PlanMoveViaGeometric (const JointVector _*currentJoints*, std::vector< CurveParameter > *viaPoints*, double *moveTime*)

Generates a Via Trajectory through multiple End-Effector positions using geometric-based constraints. This method will generate a trajectory through a number of target End-Effector positions with defined geometric constraints.

Parameters:

currentJoints This is the current position of the manipulator in Radians.
viaPoints A vector of CurveParameter objects defining the desired geometric constraints.
moveTime The move time to complete the motion.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetAccelerationScale (double *scale*)

Use this to set the Acceleration Scale for the **MotionPlanner**. This value sets the of max hardware accelerations (as set in the constructor) to use for planning trajectories. This applies to the joint limits for both point-to-point and teleoperation motions. For Cartesian motions, this will affect the hand acceleration limits provided in the constructor that are used for jogging.

Parameters:

scale The desired acceleration scale. This value must be greater than 0 and less than or equal to 1. A value of 1 will use the maximum hardware acceleration in trajectory planning. The default value is 0.5.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetBasePose (const Xform & basePose) [inline]

Use this to set the base pose of the manipulator.

Parameters:

basePose The desired manipulator base pose.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetCartesianControlMode (CartesianCoordinateMode _*coordMode*) [inline]

Use this method to set the Cartesian Coordinate mode for teleoperation. This method can be used to switch between controlled the EEF in World coordinates or Tool coordinates for teleoperation using the **PlanMoveJogCartesian()** method. Note: this method will not change the way point-to-point moves are performed.

Parameters:

_coordMode This can be set to World or Tool.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetCycleRate (double *rate*) [inline]

Use this to set the cycle rate of the manipulator controller.

Parameters:

rate The desired cycle rate in hz. The default value is 100. This value can be changed anytime the trajectory status is Inactive.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetJointPosition (const JointVector & *joints*) [inline]

Use this to set the current joint/Cartesian states of the robot. This method will update all internal kinematics using the provided joint configuration. For revolute joints, the values should be in radians. For prismatic joints, the values should be in the same units as the DH parameters.

Parameters:

joints The desired manipulator joint configuration.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetRampTime (double *rampTime*)

Use this method to set the ramp time for Cartesian motions. This method sets the of the trajectory time to use for the acceleration/deceleration motions for Cartesian . As this value increases, the start-up/slow-down will become smoother but the coast velocity will increase.

Parameters:

rampTime Valid range is 0-0.5. The default setting is 0.15.

See also:

SetTrajectoryShape()

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetSpeedScale (double *scale*) [inline]

Use this to set the Speed Scale for the **MotionPlanner**. This method can be used to slow down the motions for debugging/testing purposes. When the speed scale is set, all subsequent motions (both point-to-point and teleoperation) will be scaled slower based on this value. For example, if the speed scale is set to 0.5, all subsequent motions will execute in

exactly twice the time. Note: this value works on top of the velocity/acceleration scales set in SetVelocityScale(float scale) and SetAccelerationScale(float scale). It is mainly designed for testing new trajectories at slower, safer speeds.

Parameters:

scale The desired speed scale. This value must be greater than 0 and less than or equal to 1. A value of 1 represents a full-speed motion.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetToolPose (const Xform & *toolPose*) [inline]

Use this to set the tool pose of the manipulator.

Parameters:

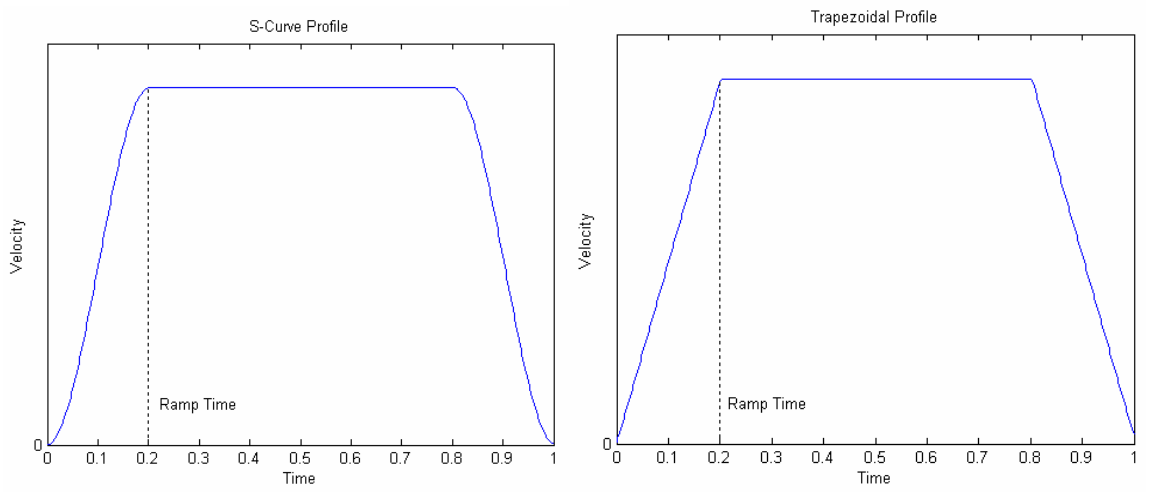
toolPose The desired manipulator tool pose.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetTrajectoryShape (TrajectoryShapeType *shape*)

Use this method to switch between Trapezoidal and S-Curve velocity profiles. This method changes the shape of the velocity profile during the acceleration period. Trapezoid uses a constant acceleration profile while SCurve uses a smoother acceleration profile. For joint interpolated motions, the amount of time used to acceleration/deceleration is based on provided limits data. For Cartesian interpolated motions, the amount of time used for acceleration/deceleration can be set using the **SetRampTime()** method.



Parameters:

shape This can be set to Trapezoid or SCurve. The default setting is Trapezoid.

See also:

SetRampTime()

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::SetVelocityScale (double *scale*)

Use this to set the Velocity Scale for the **MotionPlanner**. This value sets the of max hardware velocities (as set in the constructor) to use for planning trajectories. This applies to the joint limits for both point-to-point and teleoperation motions. For Cartesian motions, this will affect the hand velocity limits provided in the constructor that are used for jogging.

Parameters:

scale The desired velocity scale. This value must be greater than 0 and less than or equal to 1. A value of 1 will use the maximum hardware velocity in trajectory planning. The default value is 0.95.

Returns:

True if no error. False if an error. Call GetError() for detailed error information.

bool OSCAR::MotionPlanner::Stop (const JointVector & *currentJoints*, const JointVector & *currentVelocity*, bool *fastest* = true)

Use this method to stop the manipulator motion. This method will decelerate each individual axis to a velocity of 0. If called during a Cartesian move, the robot will not continue in a straight line motion. If called during a jogged move, the robot will stop its motion and return a TrajectoryComplete.

Parameters:

currentJoints The current joint state of the manipulator in Radians.
currentVelocity The current velocity state of the manipulator in Rad/s.
fastest If set to true, the manipulator will ignore the Speed Scale value and stop the manipulator as fast as possible. If set set to false, the manipulator will coast to a slower stop if the Speed Scale value is less than 1.

Returns:

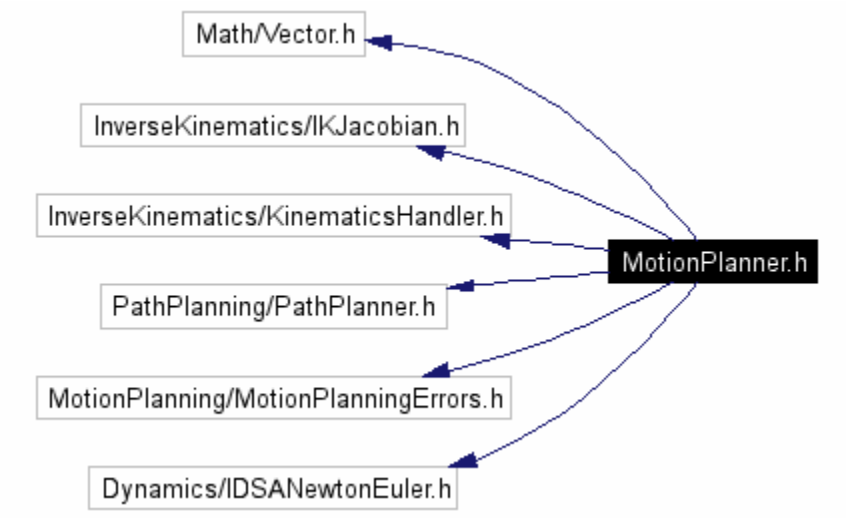
True if no error. False if an error. Call GetError() for detailed error information.

Motion Planner File Documentation

MotionPlanner.h File Reference

```
#include "Math/Vector.h"
#include "InverseKinematics/IKJacobian.h"
#include "InverseKinematics/KinematicsHandler.h"
#include "PathPlanning/PathPlanner.h"
#include "MotionPlanning/MotionPlanningErrors.h"
#include "Dynamics/IDSANewtonEuler.h"
```

Include dependency graph for MotionPlanner.h:



Namespaces

- namespace **OSCAR**

Enumeration Type Documentation

enum MPTrajectoryType

Enumeration values:

JointInterpolated
CartesianInterpolated
JointInterpOrientation

enum MPViaType

Enumeration values:

FlyBy
FlyThrough

REFERENCES

[1] Abhyankar, S. S, and Bajaj, C. L., "Automatic Parameterization of Rational Curves and Surfaces III: Algebraic Plane Curves", ACM Transactions on Graphics, Vol. 5, 1988.

[2] Abhyankar, S. S, and Bajaj, C. L., "Automatic Parameterization of Rational Curves and Surfaces IV: Algebraic Space Curves", ACM Transactions on Graphics, Vol. 8, No. 4, October 1989.

[3] Angeles, J., Rojas, A., and Lopez-Cajun, C. S., "Trajectory Planning in Robotics Continuous-Path Applications", IEEE Journal of Robotics and Automation, August 1988.

[4] Bahr, B., and Xiao, X., and Krishnan, K., "A Real-Time Scheme of Cubic Parametric Curve Interpolations for CNC Systems", Computers In Industry, Vol.45, 2001.

[5] Bajaj, C. L. and Xu, G., "A-splines: local interpolation and approximation using G^k-continuous piecewise real algebraic curves", Computer Aided Geometric Design, 18:3(2001), 149-173.

[6] Bajaj, C. L. and Xu, G., "Regular algebraic curve segments (III) – applications in interactive design and fitting", Computer Aided Geometric Design, 16:6(1999), 557-578.

[7] Barsky, B. A. and Beatty, J. C., "Local Control of Bias and Tension in Beta-splines", ACM Transactions on Graphics, Vol. 2, No. 2, April 1983.

[8] Barsky, B. A and DeRose, T. D., "Geometric Continuity of Parametric Curves: Three Equivalent Characterizations", IEEE Computer Graphics and Applications, v.9 n.6, p.60-68, November 1989.

[9] Barsky, B. A and DeRose, T. D., "Geometric Continuity of Parametric Curves: Constructions of Geometrically Continuous Splines", IEEE Computer Graphics and Applications, v.10 n.1, 1990.

[10] Barsky, B.A., "Parametric Bernstein/Bezier Curves and Tensor Product Surfaces," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-90-571, Aug. 1990.

[11] Bishop, R. L., "There is more than one way to frame a curve." *Amer. Math. Monthly*, 82(3):246-251, March 1975.

[12] Chen, H., Sheng, W., Ning, X., et al., "Automated Robot Trajectory Planning for Spray Painting of Free-Form Surfaces in Automative Manufacturing, " Proceedings of IEEE International Conference on Robotics and Automation, 2002.

[13] Craig, J. J., 1989, Introduction to Robotics, Addison Wesley, Massachusetts.

[14] DeRose, T. D. and Barsky, B. A., "Geometric Continuity, Shape Parameters, and Geometric Constructions for Catmull-Rom Splines", ACM Transactions on Graphics, Vol. 7, No. 1, January 1988.

[15] Dyllong, E., and Visioli, A., "Planning and Real-Time Modification of a Trajectory Using Spline Techniques", Robotica, Vol. 21, 2003.

[16] Frost, P., 1973, An Elementary Trestise on Curve Tracing, Fifth Edition, Chelsea Publishing Company, New York.

[17] Gallier, J., 2000, Curves and Surfaces in Geometric Modeling: Theory and Algorithms, Morgan Kaufmann Publishers, San Francisco, California.

[18] Griffiths, P. A., 1989, Introduction to Algebraic Curves, American Mathematics Society, Providence, RI.

[19] Hanson, Andrew, "Parallel Transport Approach to Curve Framing", Indiana University Computer Science Department, 1995.

[20] Hanson, Andrew, "Quaternion Frenet Frames", Technical Report 407, Indiana University Computer Science Department, 1994.

[21] Hanson, Andrew, Visualizing Quaternions, Morgan-Kaufmann/Elsevier, 2006.

[22] Hofmeyer, M. E. and Barsky, B., "Rational Continuity: Parametric, Geometric, and Frenet Frame Continuity of Rational Curves", ACM Trans. on Graphics, vol 8, 4, 1989.

[23] Kapoor, C. and Tesar, D., 1996, "A Reusable Operational Software Architecture for Advanced Robotics," Ph. D. Dissertation, University of Texas at Austin.

[24] Kapoor, C. and Tesar, D. 1998, "A Reusable Operation Architecture for Advanced Robotics," *Proceedings of the Twelfth CISM-IFToMM Symposium on the Theory and Practice of Robots and Manipulators.*

[25] Kim, J.-H., Ryuh, B.-S., and Pennock, G. R., "Development of a trajectory generation method for a five-axis NC machine", Mechanism and Machine Theory, Vol. 36, 2001.

[26] Kreyszig, Erwin, Differential Geometry, New York, Dover Publications, 1991.

[27] Lambert, J. M., Mantegh, I., and Perron, C., "3D Path Planning and Real-Time Simulation for Robot Manipulators with Applications to Aerospace Manufacturing", Proceedings of DETC 2004.

[28] Lin, C-S., Chang, P-R.; Luh, J.Y.S., "Formulation and optimization of cubic polynomial joint trajectories for mechanical manipulators", IEEE Conference on Design and Control, 1982.

[29] Lloyd, J. and Hayward, V., "Real-Time Trajectory Generation Using Blend Functions", IEEE International Conference on Robotics and Automation, 1991.

[30] Manocha, D., "Regular Curves and Proper Parameterizations", Proceedings of the international symposium on Symbolic and algebraic computation, 1990.

[31] Manocha, D. and Canny, J., "Detecting Cusps and Inflection Points in Curves", Computer Aided Geometric Design, 1992.

[32] March, P. S., "Criteria-Based Path Planning", MS Thesis, The University of Texas at Austin, 2004.

[33] Mujtaba, M. S., "Discussion of Trajectory Calculation Methods", Stanford University, Artificial Intelligence Laboratory, AIM 285.4, 1977.

[34] Paul, R.P., "Manipulator Cartesian path control," IEEE Transactions on Systems, Man, and Cybernetics, Nov. 1979.

[35] Pauluszny, M., and Patterson, R. R., "Geometric Control of G2-cubic A-splines", Computer Aided Geometric Design, Vol. 15, 1998.

[36] Pfeiffer, F, and Johanni, R., "A Concept for Manipulator Trajectory Planning", IEEE Journal of Robotics and Automation, April 1987.

[37] Plessis, L. J. and Snyman, J. A., "Trajectory Planning through Interpolation by Overlapping Cubic Arcs and Cubic Splines", Internation Journal for Numerical Methods in Engineering, Vol. 57, 2003.

[38] Rajan, Ratheesh. "Foundation Studies for an Alternate Approach to Motion Planning of Dynamic Systems", MS Thesis, 2001.

[39] Ryuh, B. S. and Pennock, G. R., "Accurate Motion of a Robot End-Effector using the Curvature Theory of Ruled Surfaces", ASME Journal Mech., Transm., Autom. Des., Vol. 110, 1986.

[40] Samuel, A. E., and Burvill, C. R., "Tracing Surfaces with a Robot Manipulator", Proceedings of IEEE International Conference on Advanced Robotics, 1991.

[41] Shoemake, Ken. "Animating Rotation with Quaternion Curves", ACM SIGGRAPH 19, 3, 245-254.

[42] Shin, K., and McKay, N. D., "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints", ", IEEE Transactions on Automatic Control, 1985.

[43] Shin, K., and McKay, N. D., "Selection o f Near-Minimum Time Geometric Paths for Robotics Manipulators", IEEE Transactions on Automatic Control, 1986.

[44] Siedel, H.-P., "Polar Forms for Geometrically Continuous Spline Curves of Arbitrary Degree", ACM Transactions on Graphics (TOG), v.12 n.1, p.1-34, Jan. 1993.

[45] Simon, D., and Isik C., "Efficient Cartesian Path Approximation for Robots Using Trigonometric Splines", Proceedings of the American Control Conference, June 1994.

[46] Suh, S.-H., Woo, I.-K., and Noh, S.-K., "Development of An Automatic Trajectory Planning System (ATPS) for Spray Painting Robots", Proceedings of IEEE International Conference on Robotics and Automation, 1991.

[47] Tesar, D., The Generalized Concept of Three Multiply Separated Points in Coplanar Motion. *J. Mechanisms*, 1967.

[48] Tesar, D., The Generalized Concept of Four Multiply Separated Points in Coplanar Motion. *J. Mechanisms*, 1967.

[49] Tesar, D. and Sparks, J. W., The Generalized Concept of Five Multiply Separated Points in Coplanar Motion. *J. Mechanisms*, 1968.

[50] Tesar, D., and Matthew, G., "The Dynamic Synthesis, Analysis, and Design of Modeled Cam Systems," Lexington Books, D. C. Heath & Company, 1976.

[51] Thomas, M. and Tesar, D. 1982 "Dynamic Modeling of Serial Manipulator Arms," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 102, pp. 218-228.

[52] Thompson, S.E.; Patel, R.V., "Formulation of joint trajectories for industrial robots using B-splines." IEEE Transactions on Industrial Electronics, 1987.

[53] Ting, K.-L., Zhang, Y., and Bunduwongse, R., "Characterization and Coordination of Point-line Trajectories", ASME Journal of Mechanical Design, Vol. 127, May 2005.

[54] Volpe, R., "Task Space Velocity Blending for Real-Time Trajectory Generation", IEEE Interation Conference on Robotics and Automation, 1993.

[55] Walker, R. J., Algebraic Curves, Princeton University Press, 1950.

[56] Watanabe, K., "Application of Natural Equations to Synthesis of Path Generating Mechanisms", Mechanism and Machine Theory, Vol. 27, 1992.

[57] Wu, C.-H., and Jou, C.-C., "Design of a Controlled Spatial Curve Trajectory for Robot Manipulators", Proceedings of IEEE 27th Conference on Decision and Control, 1988.

[58] Wu, C.-H., and Jou, C.-C., "Planning and Control of Robot Orientional Path", IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, No. 5, 1989.

[59] Zha, X. F., "A New Approach to Generation of Ruled Surfaces and its Applications in Engineering", International Journal of Advanced Manufacturing Technology, 1997.

[60] Zha, X. F., "Optimal Pose Trajectory Planning for Robot Manipulators", Mechanism and Machine Theory 37, 2002.

[61] Zha, X. F., and Chen, X. Q., "Trajectory Coordination Planning and Control for Robot Manipulators in Automated Material Handling and Processing", International Journal of Advanced Manufacturing Technology, 2004.

[62] Zha. X. F. and Du, H., "Generation and Simulation of Robot Trajectory in a Virtual CAD-Based Off-Line Programming Environment", International Journal of Advanced Manufacturing Technology 17, 2001.

[63] Zhang, Q. G., and Greenway, R. B., “Development and implementation of a NURBS curve motion interpolator”, Robotics and Computer-Integrated Manufacture, Vol. 14, 1998.

VITA

Peter Setterlund March was born in Worcester, Massachusetts on February 21st , 1978. After graduating from Maryville High School in Maryville, Tennessee, he enrolled in the Georgia Institute of Technology in Atlanta, GA. He received his Bachelors of Science in Mechanical Engineering in August, 2001. He then enrolled in the graduate program at the University of Texas at Austin where he accepted a Graduate Research Assistantship with the Robotics Research Group. In August 2004, he completed his Masters of Science in Engineering at The University of Texas at Austin.

Permanent Address: 3205 Knobdale Road
Nashville, Tennessee 37214

This thesis was typed by the author.