TRAJECTORY PLANNING ALONG CONTINUOUS PATHS SUBJECT TO PROCESS CONSTRAINTS

By

Dan Li

B.A.Sc. (Mechanical Engineering) TsingHua University, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

December 1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Mechanical Engineering

The University of British Columbia
Vancouver, Canada

Date Dec. 29. 1993

DE-6 (2/88)

Abstract

This thesis presents a new trajectory planning algorithm for planning safe and smooth trajectories of a robot tool moving along a specified curve subject to application constraints. The application constraints include geometric constraints (allowable orientation and position of the tool) and motion constraints (joint limits, robot configuration inversions and collision avoidance). The algorithm, which is based on a configuration space approach for robot motion planning, addresses the problem of motion planning along a curve rather than point-to-point motion associated with pick and place operation. Configuration spaces are searched to assess the locations of the obstacles in the vicinity of the curve and the curvature of the curve in the work space. The trajectory is planned by searching between path segments connecting successive curve points, generating the path along a curve linking the path segments, and selecting the optimal configurations along the path to construct the final trajectory. If more than one paths exist, an A^* search is employed to optimize the path. The selected path specifies a sequence of tool configuration ranges, and each configuration produces several inverse kinematic solutions. A second A^* search is used to select inverse kinematic solutions, which avoid major changes of robot configuration (i.e. joint inversion). Since the algorithm searches the allowable tool orientation range, the optimal trajectory which maximizes process quality is obtained. The trajectory allows the tool to translate and rotate simultaneously without collision, and the tool remains within the allowable orientation range and tracks the desired curve within a given tolerance. Since the step size between curve points is varied dynamically as the search proceeds, the path is planned based on a minimum number of curve points, resulting in fast path planning. A smaller step size is used if the curvature of the curve

or the likelihood of collision is high. The primary advantage of the algorithm is not only in finding a collision free path but in planning a trajectory which satisfies application constraints. It is felt that this algorithm provides a high quality process which is difficult to achieve with manual robot programming.

An off-line simulation program, TRAJPLAN, has been developed based on this algorithm. After the user inputs the application specifications, selects the curves to be processed and the environment objects of interest, TRAJPLAN automatically plans the trajectory of robot tool and generates the required set of sequential robot joint angles. TRAJPLAN plans the motion with different precision depending on the requirements of the application.

This algorithm has been demonstrated for robotic fish butchering and welding, and the issues related to these applications are discussed in this thesis. As an example of a typical problem, the system automatically plans the trajectory of a robot tool moving along a 3-D curve with 31 points and surrounded by 5 obstacles in 1'20". The algorithm is suitable for a variety of robot applications which require the robot tool to move along a curve, keep a certain orientation and position relative to the curve, and avoid collision with the environment.

Table of Contents

List of Figures

List of Symbols

α the angle rotating around y axis;

β the angle rotating around z axis;

γ the angle rotating around tool axis;

α_o the α angle of optimal orientation;

β_o the β angle of optimal orientation;

γ_o the γ angle of optimal orientation;

α_{min} the minimum α angle in *Local C-space*;

α_{max} the maximum α angle in *Local C-space*;

β_{min} the minimum β angle in *Local C-space*;

β_{max} the maximum β angle in *Local C-space*;

x, y, z the coordination of a curve point.

p_i the *ith* curve point.

m_i the *ith* midpoint.

d_{min} the minimum distance between robot tool tip and curve.

d_{max} the maximum distance between robot tool tip and curve.

List of Key Words

Local C-space	the local configuration space.
Entire C-space	the entire configuration space.
local θ region	the region of *Local C-space* in α-β plane.
local γ range	the range of *Local C-space* in γ axis.

θ swept volume	the tool swept volume corresponding to *θ region.*
γ swept volume	the tool swept volume corresponding to *γ range.*
cell	the space in *Local C-space.*
free cell	the *cell* associated with a collision free swept volume.
θ region	the projective region of *cell* on $\alpha - \beta$ plane.
γ range	the projective range of *cell* on γ axis.
path list	the linked list containing all of the potential paths along the curve.
minimum step size	the predetermined minimum step size between successive curve points.
track point	the point on the track of robot tool tip.
intermediate curve point	the intermediate point selected from the curve segment between successive curve points.
intermediate track point	the intermediate point selected from the track segment between successive track points.
intermediate number	the number of intermediate points.
overlap-cell$_{i,i+1}$	the overlapping space of *Local C-spaces* at point p_i and p_{i+1}.
overlap-region$_{i,i+1}$	the projection of *overlap-cell*$_{i,i+1}$ on α-β plane.
free-cell$_{i,i+1}$	the space inside *overlap-cell*$_{i,i+1}$, associated with a collision free swept volume.
free-region$_{i,i+1}$	the projection of *free-cell*$_{i,i+1}$ on α-β plane.
free-range$_{i,i+1}$	the projection of *free-cell*$_{i,i+1}$ on γ axis.
shrink step	the shrink amount at each shrink.
shrink region	the region left in *overlap-region*$_{i,i+1}$ after each shrink.
θ-trans swept volume	the tool head swept volume corresponding to *overlap-region*$_{i,i+1}$.
γ-trans swept volume	the tool handle swept volume corresponding to *overlap-region*$_{i,i+1}$.
intersection orientations	the orientations corresponding to the intersection points of two *θ regions*

shrink swept volume	the swept volume corresponding to *shrink region*.
PathCost	the total cost of going through a path.
TrajCost	the total cost of going through a trajectory.

Acknowledgment

I would like to thank my supervisor, Dr. R. G. Gosine, for his guidance in developing the motion planning algorithm. This work has been undertaken with the support of the NSERC Chair in Fish Processing Automation. Discussion with B. Konesky, Dr. D. Cherchas, and Dr. F. Sassani have been of assistance in integrating this algorithm with AutoRoboWeld, an off-line programming system for welding robots. Thanks also to G. Wallace and M. Lookman of A & R Metals for useful feedback regarding motion planning requirements for welding applications.

Chapter 1

Introduction

1.1 Motivation

When using a teach pendant to program robots for contour following applications such as welding, cutting or painting, the operator must manually manipulate a robot tool along a desired curve. It is generally agreed that this process is slow and demanding on the user, and an automatic process to aid in motion planning is necessary. A typical welding robot workcell is illustrated in Figure 1.1. Safe and smooth motion of the robot and the tool is expected, and good process quality is significant in such applications. Using a robot instead of a human arm to handle the tool presents a significant planning problem. An intelligent agent is required to plan and control the motion of the robot and the tool in order to achieve good process quality. The primary contribution of this thesis is the development of a planning strategy that considers the following requirements:

1. The tip of the robot tool must move along the curve and within a certain distance from the curve.

2. The optimal orientation, as specified by the process, is relative to the tangent of the curve and the normals of two surfaces at the each side of the curve as shown in Figure 1.2. This orientation varies with the changes to the curve and the supporting surfaces. The robot tool has to maintain the optimal orientation within a given tolerance.

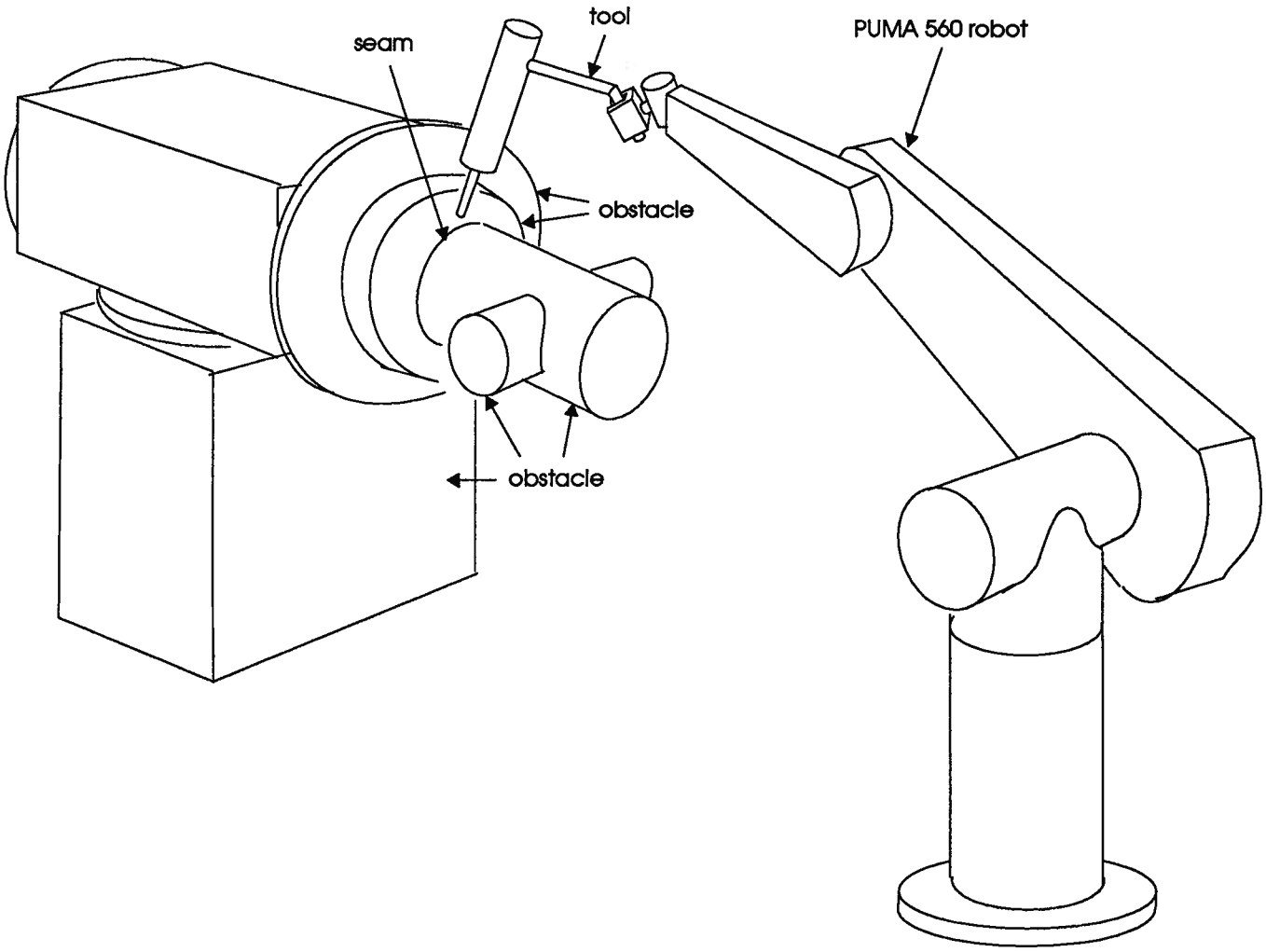

Figure 1.1: The robot welding workcell

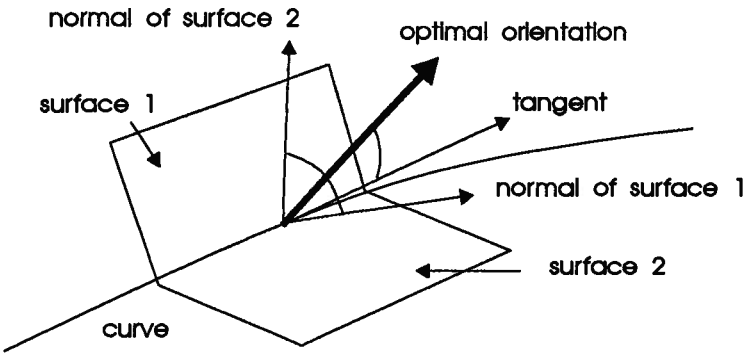

Figure 1.2: The optimal orientation of robot tool

3. The robot tool must avoid collisions with the curve and the obstacles around the curve.

4. The position and orientation of robot tool result in a set of robot joint angles which do not violate the joint limits.

5. A significant change of robot configuration (i.e. arm inversion) must be avoided .

6. Smooth motion of the robot and tool is expected.

In the above requirements, the tool tip must trace the curve within a given tolerance, and the tool must maintain an orientation close to the optimal one within a given tolerance. These tolerances are specified by the application in order to maintain acceptable process quality.

Collision avoidance is a basic requirement for tool motion. Within the allowable and collision free orientation range, some tool orientations may lead to invalid robot inverse kinematics solution (the joint angles in the solution violate the joint limits). It is difficult for a human operator to select orientations which lead to valid solutions.

The motion planner must maintain a path history in order to provide smooth motion of the tool. Changes of robot configuration will also reduce process quality because of

discontinuous robot motion. The robot configuration, however, may have to change as the tool travels a curve. It is difficult for the human programmer to pre-judge such changes and prevent them by selecting alternate robot configuration at the beginning of process.

It is impossible for the human programmer to consider all of these constraints simultaneously while using the teach pendant to program the robot. Programming the robot motion becomes difficult and the process quality is compromised.

The subject of this research is to to plan the motion of the robot tool automatically and meet the above requirements.

1.2 Midpoint Algorithm: a Novel Approach to Motion Planning

A more intelligent robot tool trajectory planner is required and this thesis proposes such an algorithm, the Midpoint Algorithm, to plan the trajectory of robot tool tracing a curve subject to process constraints. A safe and smooth robot tool trajectory is selected by the algorithm, which is based on the configuration space approach [Loza 83], to provide high process quality. Since a curve is represented by discrete points, the motion of the tool along the curve links the motion between successive curve points. By studying the configuration spaces at successive curve points and the relationship between the configuration spaces, the algorithm assesses the locations of obstacles in the vicinity of the curve and the curvature of the curve in work space, finds the collision free path and plans an optimal trajectory for the tool. If more than one paths exists, an A^* search optimizes the path. A polyline approximating the curve is used as a reference for the tool tip and the algorithm varies the step size between points on the polyline according to the condition of the curve and the environment. A large step size is used to speed up the path planning if the curvature of the curve or the likelihood of collision is low, and

a small step size is used for more intricate robot motion if the curvature of the curve or the likelihood of collision is high. In order to assure that the orientation of the robot tool is within the allowable orientation range as it moves along the curve, exploration of the space around the optimal orientation at each curve point is necessary. The configuration space represents the interaction of the robot and environment at each curve point. A path is found through the configuration space using an A^* search for possible solutions.

An off-line simulation program, TRAJPLAN, implements the algorithm in order to demonstrate the application potential of the work. Examples based on the PUMA 560 and CRS A460 robots have been developed and tested on an real robot.

1.3 Outline of Thesis Content

The remainder of this thesis is arranged as follows. Chapter 2 reviews previous related work in the areas of robot motion planning based on the configuration space, path planning for robotic tools, curve representation, and search techniques. Chapter 3 details the basic concepts behind the Midpoint Algorithm, while Chapter 4 provides details of the Midpoint Algorithm. Chapter 5 describes the simulation program TRAJPLAN and its application to typical planning problems. Finally, Chapter 6 presents conclusions for this work and suggestions for future work.

Chapter 2

Literature Review

Two general path planning problems are involved in the motion planning of a robot tool tracing the specified curves: collision avoidance and the motion along the curve.

2.1 A Basic Motion Planning Problem: Collision Avoidance

Robot motion planning has been extensively studied and good survey articles include [Cann 88] [Lato 91] [Shar 89]. One of the basic problems in robot motion planning is to find a collision free path for the robot. This problem is to search for a path from an initial position and orientation to a goal position and orientation of robot. The path specifies a continuous sequence of positions and orientations of the robot avoiding contact with the obstacles.

If the obstacles are static and the locations of obstacles in the work space are available, a popular robot path planning method is the *configuration space* approach. This approach is initialized by Udupa [Udup 77] who proposed the motion planning amidst obstacles and described the idea of using an appropriate space and shrinking the robot to a point in the space. This idea is exploited by Lozano-Pérez and Wesley [Loza 79]. They proposed a path planning algorithm for polygonal and polyhedral robots and obstacles without rotation. Lozano-Pérez [Loza 81] extended the algorithm proposed in [Loza 79] and used the notion of *configuration space*. Some methods based on this approach, such as *roadmap*, *cell decomposition* and *potential field*, are discussed in Section 2.2.

Dynamic motion planning [Cann 88] [Lato 91] [Reif 85] deals with moving obstacles

in work space. A continuous function of time is used to specify the robot configuration at each instant. In effect, a dimension representing time is added to the conventional *configuration space.* In this configuration-time space, the robot is shrunk to a point moving among stationary obstacles. Since no object can move back in time, the path found is monatomic in time. Constraints on robot velocity and acceleration have to be considered.

In the presence of multiple robots in the work space, one of the motion constraints is that the robots cannot occupy the same space at the same time. *Centralized planning* [Schw 83] and [Tour 86] generates the composite configuration space of the robots and plans the coordinated paths in the space. Some general path planning methods such as *cell decomposition* and *potential field* can be used to plan a path in the composite configuration space. Another approach to this problem is the *decoupled planning* method which plans the motion of each robot independently of the other robots and then considers the interactions among the paths. There are two decoupled planning approaches: *prioritized planning* and *path coordination. Prioritized planning* [Buck 89] considers the motions of all robots, one robot at a time. The path of a robot is generated in a configuration-time space of the robot at each iteration, avoiding the collisions with both the obstacles and the other robots. The motion of the robot is planned as if the robot was moving among stationary obstacles and some moving obstacles (other robots). To arrange the order of robots to be planned, a priority is assigned to each robot. Buckley [Buck 89] assigned priorities to the robots which move in straight line from their initial configurations to their goal configurations. *Path coordination* [Odon 89] is applicable when the planning problem involves only two robots. A free path for each of the two robots is generated independently of the other robot. The two free paths are coordinated in order to avoid the collision between the robots.

Another general approach to solving path planning is to search the free space directly

without first transforming the problem to *configuration space*. Kambhampati and Davis [Kamb 86] proposed a path planning approach based on a Quadtree representation. They described the hierarchical path-searching methods which make use of multiresolution representation to speed up the path planning process considerably. Canny and Lin [Cann 90] presented a planning algorithm which traces out curves of maximal clearance from obstacles and creates a one-dimensional roadmap of the free space of a robot. Their method takes advantage of the fact that the closest features (vertex, edge, or face) change only infrequently as the objects move along finely discreted paths. Gupta [Gupt 90] described a method of planning the motion of each link of a manipulator successively, initializing from the base link. The n-D problem for an n-link manipulator arm transforms to one 1-D and $n-1$ 2-D planning problems. This 2-D motion planning problem is to plan the motion of a single link which has one end moving along a fixed path determined by the motion of the previous links. An interesting collision avoidance algorithm is reported by Shaffer and Herb [Shaf 92]. They described a data structure and data structure update algorithm for a real-time collision avoidance safety system. The N-objects octree is used to index a collection of 3-D primitive solids which make up the robots and obstacles in work space. As robots move, the octree is updated and the octree nodes recursively decompose 3-D space into eight equal cubic octants until each octant contains less than a predetermined number N of primitives. Since a given primitive does not change which octree nodes it is in during most update cycles, the octree is rarely modified. The system provides the information about possible collision.

2.2 Path Planning Methods Based on the Configuration Space Approach

The *configuration space* method [Udup 77] [Loza 79] [Loza 81] [Loza 83] [Loza 87] shrinks the robot as a point in robot *configuration space* and maps the obstacles in this space.

The motion planning of a point instead of a dimensioned object is more explicit. The *configuration space* represents the robot and obstacles in a coordinate system defined by the joints of the robot. There exists a large number of methods of solving the motion planning problem based on the concept of *configuration space*. Three general approaches are *roadmap*, *cell decomposition*, and *potential field* [Lato 91].

The *roadmap* approach represents the free space in the form of a network of one-dimensional curves, called *roadmap*, which is used as a set of standardized paths. Path planning becomes a search for three path segments: the path segment connecting the initial configuration to the *roadmap*, the path segment contained in the *roadmap*, and the path segment connecting the *roadmap* to the goal configuration. The *visibility graph* method, *Voronoi diagram* method and *freeway* method are all based on this idea. The *visibility graph* method [Nils 69] is suitable to a two-dimensional *configuration space* with polygon configuration obstacles. The nodes of the graph are the initial and goal configurations and the vertices of the configuration obstacles. Two nodes are connected by a straight line segment if the line does not intersect the interior of the configuration obstacles. The path is found by searching for the shortest path in the graph. The *Voronoi diagram* method [Taka 89] generates the Voronoi diagram which is in the free regions of *configuration space*. The Voronoi diagram is the locus of points which are equidistant from two or more obstacle boundaries including the work space boundary. The advantage of this diagram is that it produces free paths which tend to maximize the clearance between the robot and the obstacles. The *freeway* method [Broo 83(a)] plans the path for a polygonal object (robot) translating and rotating in a two-dimensional space. A *freeway* is a straight linear generalized cylinder. The straight axis of the cylinder, called the spine, is the path of the reference point of the robot. A free space is represented by a *freeway* net. In a *freeway* net, if two spines intersect, the robot can transfer from one free way to the other if the ranges of free orientations of the robot along both splines have a

non-empty intersection at the intersection point. The *freeway* net is a representation of the possible motion of a robot along spines and between spines.

The *cell decomposition* method decomposes the robot free space into some simple regions, called *cells.* A graph representing the adjacency relation between the cells is constructed by extracting the cells from the free space and connecting two graph nodes if these cells are adjacent. The outcome of the graph search is a sequence of cells and a path is extracted from this sequence. There are two approaches in this area: *exact cell decomposition* and *approximate cell decomposition.* The *exact cell decomposition* method [Chaz 87] decomposes the free space into trapezoidal and triangular cells, represents the adjacency relation between the cells by constructing the connectivity graph, and searches for a path in this graph. The cells used in the *exact cell decomposition* method are required to have a simple pre-specified shape, e.g. a rectangle. Such cells do not provide an exact representation of free space. The *approximate cell decomposition* method [Loza 81] [Broo 83(b)] uses Quadtree decomposition to divide a cell into four subcells. A cell is recursively decomposed until a predetermined resolution is attained, or the cell lies completely in free space or in the configuration obstacle. The path is searched through the cells lying in free space. Since the shape of a cell is relatively insensitive to numerically approximate computations, this method is usually much easier to implement than the *exact cell decomposition* method.

In the *potential field* method [Khat 86], the robot is represented as a particle in *configuration space.* The goal configuration generates an "attractive potential" which pulls the robot toward the goal, and the configuration obstacles produce a "repulsive potential" which pushes the robot away from them. Robot moves under the influence of an artificial potential produced by the goal configuration and the configuration obstacles.

Based on the *configuration space* approach, most studies about collision avoidance focus on the trajectory of manipulators [Sing 91] [Warr 89] and mobile robots [Kamb 86]

[Taka 89] in point-to-point motion problems. There is little reported, however, on the problem of planning the robot tool motion along a 3-D curve subject to physical (obstacle) and process constraints.

2.3 Path Planning for Robot Tool Along a Continuous Curve

The principles and techniques of a high-speed spatial seam tracking system are described in [Boll 71]. A five degree of freedom seam tracer is developed, which positions a welding gun to maintain a fixed relationship to a weld line. The proper angular relationships are achieved by utilizing velocity information from three tachometers mounted on the three rectangular coordinate drives. Collision avoidance is not considered in the system, and the seam discussed in the paper is constrained to lie on a cylindrical surface.

Tomizuka, Dornfeld and Purcell [Tomi 80] discussed the characteristics of the gas metal arc welding process and the relationship between welding parameters, the desired output of the welding process, and the automation of the process. A strategy for two-axis welding torch positioning and velocity control is developed. The torch positioning and velocity control is achieved by keeping the relative position error between the seam and torch small and maintaining uniform tracking speed. In this approach, the algorithm constrains the seam to a plane and only two-axis control is available. Collision avoidance is not considered.

An algorithm for tracing a seam in real time is developed in [Khos 85]. The seam tracking control traces a curve in 3-D space and maintains proper tool orientation with respect to the surface that contains the curve. The curve lies on a surface and is discretized lengthwise. The discretization provides a piece-wise linear approximation of the curve between two adjacent sample points. The tool orientation is obtained by discretizing the surface in the vicinity of the two sample points, and this is held constant when

the tool travels between the two points. The position of the torch is specified by the coordinates of the sample points. Three unit vectors are used to describe the tool orientation relative to the base frame but the rotation about tool axis is not discussed. This may be suitable for the CYRO robot, a special robot with three revolute joints and three prismatic joints, but is not suitable for robots like the PUMA, CRS or GMF with which the rotation about the tool axis changes the position of the robot end-effector if the tool handle is held by the robot. The allowable tool orientation range and the optimal tool orientation subject to process constraints are not discussed in the paper. The sample points are used directly in seam tracking regardless of the condition of the curve and the environment around the points. This reduces the accuracy along complex curve segments and wastes time along simple curve segments.

A trajectory planner and a method for interference detection between objects in a robot welding workstation are presented in [Buch 89]. The trajectory planner searches for a kinetically feasible, interference-free robot trajectory in order to follow a specified weld path in 3-D space. The algorithm provides rapid interference detection between convex polyhedras using a steepest descent graph search. If desired, exact separating distances and directions can be found. Interference-free trajectories that do not violate kinematics constraints are found by searching possible inverse kinematic solutions for the desired welding torch trajectory. This approach only deals with tool rotation about its axis, but not the allowable tool orientation range. This paper and the associated work form the starting point for the application of the work described in this thesis.

Angeles, Rojas and Lopez-Cajun [Ange 88] discussed the angle velocity and angle acceleration of the end effector which holds a robot tool and travels along a continuous-path. The technique proposed allows the trajectory-planning engineer to specify the orientation of the end effector via the orientation of the path at a given point. The latter is defined as that of the orthonormal triad of vectors associated with the curve, namely,

its tangent, normal, and binomial vectors. The angular acceleration of the end effector requires the computation of the derivatives of the curvature and torsion with respect to the arc length. Thus the coordinate frame defined by the triad of unit tangent, normal, and binormal vector defines not only the orientation, but also the angular velocity and acceleration of a body tracking the curve with a given speed, whose relative orientation with respect to the curve remains constant, or a given function of time.

2.4 The Spline of Curve

Curve representation is an important aspect of this research, and many algorithms for curve splines have been reported. A good survey of the algorithms is found in [Fole 91]. Cubic polynomials are most often used since lower-degree polynomials give too little flexibility in controlling the shape of the curve, and higher-degree polynomials can introduce unwanted wiggles and also require more computation. The spline is a piece-wise continuous curve made up of cubic polynomials. The famous B-splines method [Bars 80] [Bart 87] consists of curve segments whose polynomial coefficients depend on just a few control points and movement of a control point affects only a small part of a curve. This is called *local control.* B-splines, however, do not interpolate the control vertices. In this research, a curve representation based on some sample points must be reconstructed and the reconstructed curve has to pass through the sample points. The B-spline is not suitable in this case.

One member of the Catmull-Rom Spline [Catm 74] [Barr 89] family interpolates the points P_1 to P_{m-1} from the sequence of points P_0 to P_m. In addition, the tangent vector at point P_i is parallel to the line connecting point P_{i-1} and P_{i+1}, as shown in Figure 2.1. The Catmull-Rom Spline interpolates control points, shares many properties such as global smoothness and local control with B-spline curves, and has continuity of the

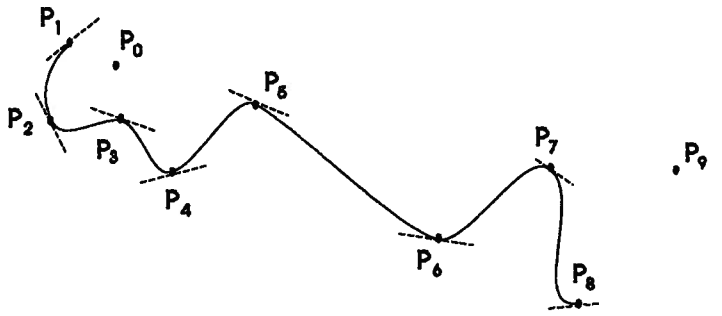

Figure 2.1: A Catmull-Rom spline

first derivative, which is required by the proposed algorithm. Given a series of points, the Catmull-Rom Spline method generates a curve smoothly to pass through them. The Catmull-Rom Spline method is suitable in this research.

2.5 Searching Path Through a Net

A further requirement of the research is an algorithm for evaluating potential tool paths and for guiding the search through path options. Procedures for finding the shortest path through a net are surveyed in [Wins 84]. The British Museum Procedure finds all possible paths and select the best from them. Since this method looks everywhere and is inefficient, it is not suitable for our path planning where search time is of concern. The Branch-and-Bound search extends the search tree in the direction of the "best" partial path. The A^* search improves the Branch-and-Bound search by including an estimate of remaining distance, combined with the dynamic-programming principle. If the estimate of remaining distance is a lower bound on the actual distance, then the A^* search achieves the optimal solution. The A^* search is used in the proposed algorithm to search multiple paths along the curve and to select the optimal trajectory from the tool configurations in the path.

2.6 Conclusion Based on Literature Review

In view of this literature survey, it is concluded that little work has been done on the problem of robot tool planning along a desired trajectory of the tip of the tool. Furthermore, conventional planning techniques have been largely concerned with point-to-point problems and did not consider application constraints associated with the required motion of the tool. The configuration space method appears promising in terms of a general approach, however, the proposed method studies the relationship between *configuration spaces* rather than searching for paths inside a single *configuration space.* For the ancillary functions, such as for effective curve representation and search procedures, existing techniques will be employed directly in this thesis.

Chapter 3

Configuration Space of Robot Tool

3.1 Configuration of Robot Tool

In this section the representation of the position and orientation of the robot tool is discussed. The foundation for this representation is a 3-D coordinate system, and Figure 3.1 illustrates the tool representation, in this case for a welding torch, at the origin of the 3-D coordinate system. Note that all objects in the environment are described with respect to this 3-D reference frame. As illustrated in Figure 3.2, the tool has six degrees of freedom, x, y, z, α, β and γ, where x, y and z are the position coordinates of the tool tip, α is the angle of rotation about the Y axis, β is the angle of rotation about the Z axis, and γ is the angle of rotation about the tool axis. The tool axis passes though the tool tip and represents the critical orientation of the tool, as shown in Figure 3.1. The configuration of the tool in the base coordinate system is determined by rotating the tool to an orientation (α,β,γ) and translating the tool from the origin to (x,y,z) while maintaining this orientation. As mentioned in Chapter 1, the six requirements as the tool travels a curve include (1) the distance between the tool tip and the curve, (2) the optimal orientation, (3) collision avoidance, (4) robot joint constraint, (5) robot arm inversion avoidance and (6) smooth motion. The distance between the tool tip and the curve point can be adjusted easily by changing the tool tip position (x,y,z). The tool orientation requirement specifies the tool orientation in the direction of the optimal orientation. The critical tool orientation is determined by the α and β angles. The initial position of tool

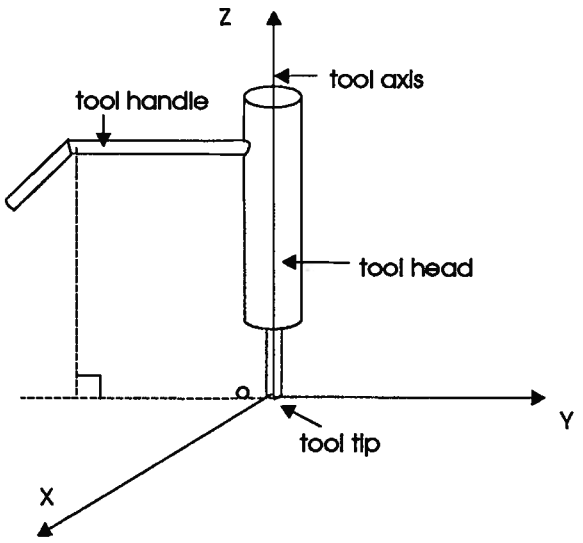

Figure 3.1: Robot tool

in Figure 3.2 (a) shows the tool orientation where α, β and γ are all equal to 0^o. The reference orientation of the γ angle is along the negative Y axis. Although it does not constitute a tool orientation constraint, the γ angle is very important in the calculation of the robot inverse kinematic solutions since a small change in γ can result in a large difference of the position and orientation of the robot end effector and the associated joint values for the robot as illustrated in Figure 3.2 (b). As discussed in Chapter 4, the γ angle is critical in the collision-free path search.

3.2 Configuration Space of Robot Tool

An alternative representation of tool orientation is the configuration space (C-space), and here two new concepts, *Entire C-space* and *Local C-space*, are introduced. The *Entire C-space* is the configuration space with its origin at α, β and γ equal to 0^o. The ranges of the α, β and γ angles in *Entire C-space* are $[0^o,180^o]$, $[0^o,360^o]$ and $[0^o,360^o]$ respectively.

The *Entire C-space* represents the complete set of tool orientations at any curve point in the work space. The α and β ranges of *Entire C-space* correspond to the sphere in

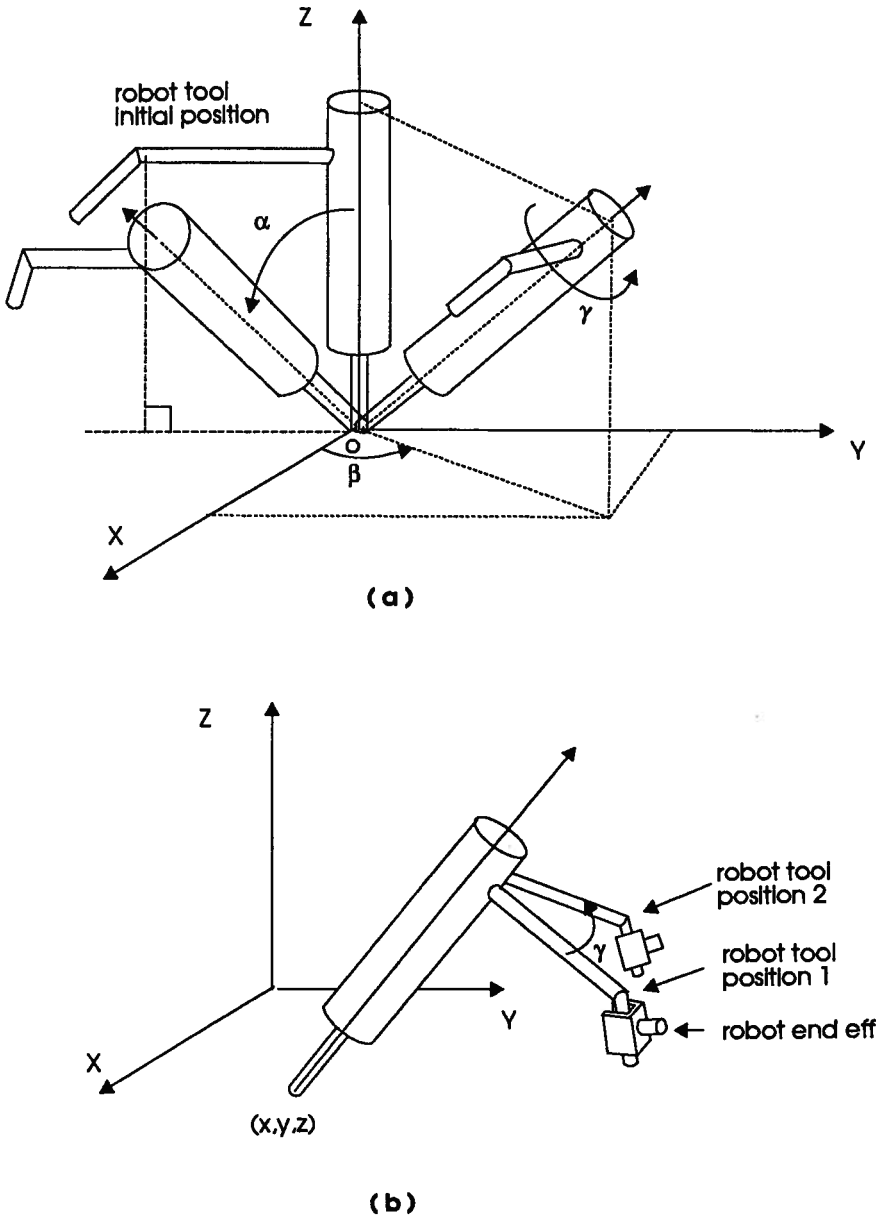

Figure 3.2: Definition of robot tool configuration

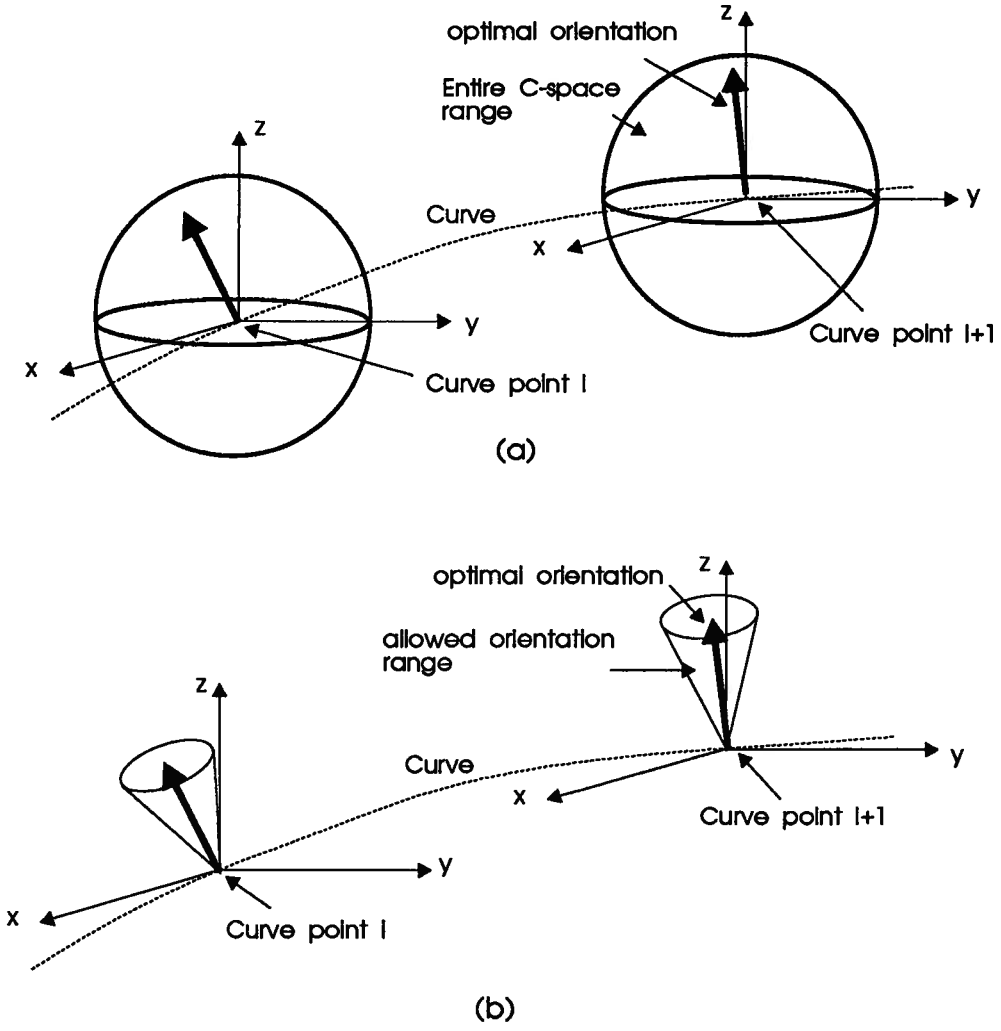

Figure 3.3: The α and β ranges of *Entire C-space* and the allowable orientation range at different curve points. (a) *Entire C-space.* (b) Allowable orientation range.

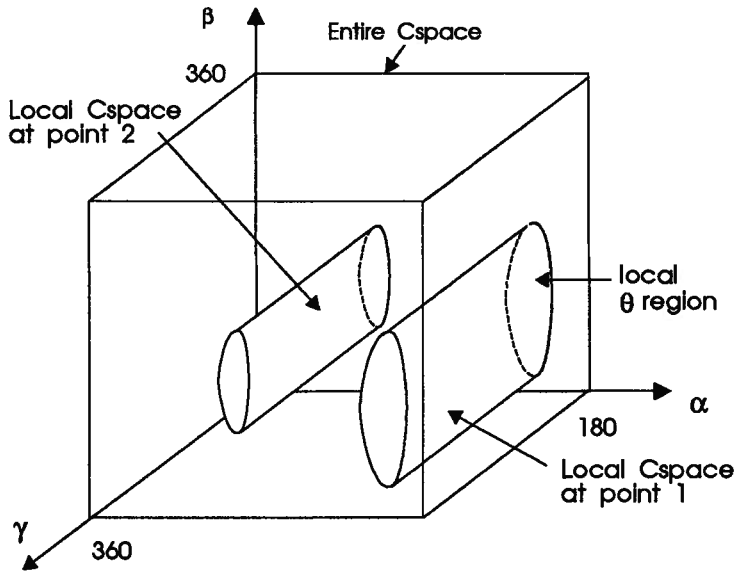

Figure 3.4: *Entire C-space* and *Local C-space*

work space as shown in Figure 3.3 (a). Using the *Entire C-space* directly in trajectory planning is not practical since a high resolution *Entire C-space* would require a large data structure, which would waste time and space as a result of creating, searching and storing much useless information in the data structure. Due to process constraints, such as allowable deviation in tool orientation from a user-specified ideal orientation, only part of the *Entire C-space* is required. To address the problem of the large *Entire C-space*, the concept of *Local C-space* is introduced.

At each curve point the optimal tool orientation depends on the process. For example, the optimal welding torch orientation may be selected by an expert welder. Similarly there will be optimal cutter orientations for an automatic butching operation. The acceptable orientations covers only a subset of the *Entire C-space*, as illustrated in Figure 3.3 (b), and only the configurations within this subset need to be considered. The α and β ranges of the *Local C-space* represent the allowable deviation from the optimal orientation for a particular application. Since the acceptable range of deviation is generally

much smaller than the range in *Entire C-space*, the *Local C-space* requires a condensed data structure in order to represent the configuration space with the same resolution as the *Entire C-space.*

With reference to Figure 3.4, the *Entire C-spaces* at different curve points have the same α and β ranges, i.e. [0,180] and [0,360] respectively. The α and β ranges of *Local C-space* are different and correspond to the different regions in the $\alpha - \beta$ plane. The projections of *Local C-space* on the $\alpha - \beta$ plane and the γ axis are called the *local θ region* and the *local γ range* respectively.

In order to visualize the approximate shape of the *local θ region*, the bounding box of the region is studied. In Figure 3.5 (a), the maximum allowable deviation of tool orientation from the optimal orientation is θ_o and the optimal orientation is (α_o, β_o). The α_{min}, α_{max}, β_{min} and β_{max} angles in Figure 3.5 (b) (c) (d) are the minimum and maximum α and β angles in the *local θ region* respectively. In Figure 3.5 (a), points 1, 2, 3 and 4 are at the boundary of the cone centered at (α_o, β_o). Figure 3.5 (b) illustrates the viewing direction in the negative optimal orientation. The line connecting point 2 to point 4 is perpendicular to the line connecting point 1 to point 3. Figure 3.5 (c) and (d) illustrate that β_{min} and β_{max} are not the β angles at point 2 and point 4, while α_{min} and α_{max} correspond to the α angles at point 1 and point 3 respectively. Note that α_{min} and α_{max} are equal to $\alpha_o - \theta_o$ and $\alpha_o + \theta_o$ respectively. The α angles at point 2 and point 4 are equal but they are not equal to α_o. The corresponding *local θ region* has an irregular shape as shown in Figure 3.5 (e).

From the above discussion, it is clear that the *local θ region* is hard to represent using a simple geometric model, such as a rectangle, circle or ellipse. With reference to Figure 3.6, equations defining the *local θ region* are derived. In the figure, the sphere represents an *Entire C-space*, which has origin o_r and radius ρ. The cone represents the *local θ region* of *Local C-space.* The intersection of optimal orientation and the sphere is

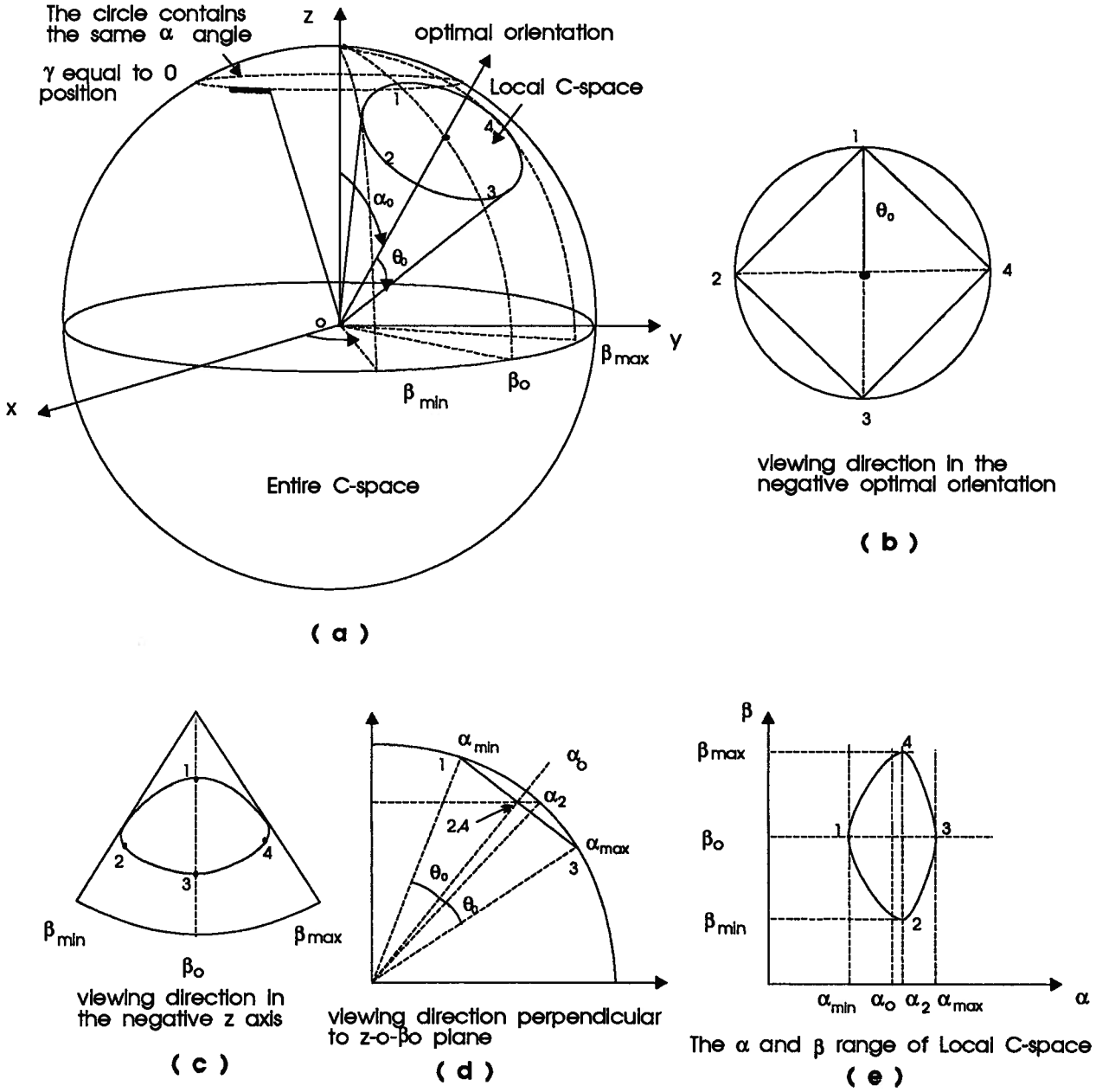

Figure 3.5: The shape of *local θ region*

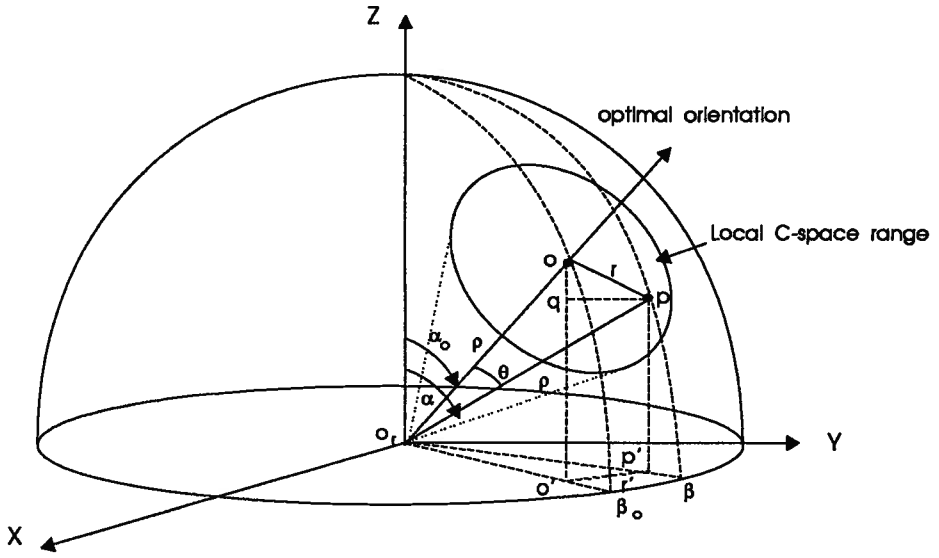

Figure 3.6: The calculation of *local θ region* of *Local C-space*

denoted as o, p is a random point within the *Local C-space* and located on the sphere, and θ is the angle between $o_r o$ and $o_r p$, where $0 \leq \theta \leq \theta_o$. The projections of o and p into X-Y plane are denoted as o' and p' respectively, while r' is the line connecting o' and p', and pq is a line parallel to $p'o'$. Given the optimal orientation (α_o, β_o) and the maximum allowable orientation deviation θ_o, the calculation of the corresponding *local θ region* is now described.

Using the cosine law in triangle $o_r op$

$$2\rho^2 - 2\rho^2 cos\theta = r^2 \tag{3.1}$$

Using the cosine law in triangle $o_r o'p'$

$$\rho^2 sin^2\alpha + \rho^2 sin^2\alpha_o - 2\rho^2 sin\alpha sin\alpha_o cos(\beta - \beta_o) = r'^2 \tag{3.2}$$

Since pq is parallel to $p'o'$ and oo' is perpendicular to $p'o'$, the triangle opq is a right triangle, and

$$r'^2 + (\rho cos\alpha_o - \rho cos\alpha)^2 = r^2 \tag{3.3}$$

Combining Equations (3.1), (3.2) and (3.3),

$$2\rho^2 - 2\rho^2 cos\theta = \rho^2 sin^2\alpha + \rho^2 sin^2\alpha_o - 2\rho^2 sin\alpha sin\alpha_o cos(\beta - \beta_o) + (\rho cos\alpha_o - \rho cos\alpha)^2 \quad (3.4)$$

which may be reduced to

$$cos\theta = cos(\alpha + \alpha_o) + [cos(\beta - \beta_o) + 1] sin\alpha sin\alpha_o \quad (3.5)$$

by the following steps,

from Equation (3.4),

$$2 - 2cos\theta = sin^2\alpha + sin^2\alpha_o - 2sin\alpha sin\alpha_o cos(\beta - \beta_o) + (cos\alpha_o - cos\alpha)^2$$

$$2 - 2cos\theta = sin^2\alpha + sin^2\alpha_o - 2sin\alpha sin\alpha_o cos(\beta - \beta_o) + cos^2\alpha_o - 2cos\alpha_o cos\alpha + cos^2\alpha$$

$$2 - 2cos\theta = 2 - 2sin\alpha sin\alpha_o cos(\beta - \beta_o) - 2cos\alpha_o cos\alpha$$

$$cos\theta = sin\alpha sin\alpha_o cos(\beta - \beta_o) + cos\alpha_o cos\alpha$$

$$\begin{aligned} cos(\theta) &= sin\alpha sin\alpha_o cos(\beta - \beta_o) + cos\alpha_o cos\alpha + sin\alpha sin\alpha_o - sin\alpha sin\alpha_o \\ &= -sin\alpha sin\alpha_o + cos\alpha_o cos\alpha + sin\alpha sin\alpha_o (cos(\beta - \beta_o) + 1) \\ &= cos(\alpha + \alpha_o) + [cos(\beta - \beta_o) + 1] sin\alpha sin\alpha_o \end{aligned}$$

Since $0 \leq \theta \leq \theta_o$,

$$1 \geq cos\theta \geq cos\theta_o$$

and from Equation (3.5), we can write,

$$1 \geq cos(\alpha + \alpha_o) + [cos(\beta - \beta_o) + 1] sin\alpha sin\alpha_o \geq cos(\theta_o)$$

$$1 - cos(\alpha + \alpha_o) \geq [cos(\beta - \beta_o) + 1] sin\alpha sin\alpha_o \geq cos(\theta_o) - cos(\alpha + \alpha_o) \quad (3.6)$$

Next, consider the left hand side of Equation (3.6). Since $1 \geq cos(\beta - \beta_o)$,

$$2 \geq cos(\beta - \beta_o) + 1$$

Since both α and α_o are in the range [0, 180], and

$$sin\alpha sin\alpha_o \geq 0$$

we can write,

$$2sin\alpha sin\alpha_o \geq [cos(\beta - \beta_o) + 1]sin\alpha sin\alpha_o \tag{3.7}$$

Since $1 \geq cos(\alpha - \alpha_o)$, we know

$$1 - cos(\alpha - \alpha_o) \geq 0$$

and

$$1 - (cos\alpha cos\alpha_o + sin\alpha sin\alpha_o) + 2sin\alpha sin\alpha_o \geq 2sin\alpha sin\alpha_o$$

or

$$1 - cos(\alpha + \alpha_o) \geq 2sin\alpha sin\alpha_o \tag{3.8}$$

Combine Equation (3.7) and Equation (3.8)

$$1 - cos(\alpha + \alpha_o) \geq 2sin\alpha sin\alpha_o \geq [cos(\beta - \beta_o) + 1]sin\alpha sin\alpha_o$$

The left hand side of Equation (3.6) is always satisfied. Now, consider the right hand side of Equation (3.6).

Since both α and α_o are in the range of [0, 180],

$$sin\alpha sin\alpha_o \geq 0$$

and

$$[cos(\beta - \beta_o) + 1]sin\alpha sin\alpha_o \geq 0 \tag{3.9}$$

First consider the case when

$$cos(\theta_o) - cos(\alpha + \alpha_o) \leq 0$$

We can write,

$$[cos(\beta - \beta_o) + 1]sin\alpha sin\alpha_o \geq cos(\theta_o) - cos(\alpha + \alpha_o)$$

The right hand side of Equation (3.6) is always satisfied, and β can be any value in the range [0, 360].

The condition

$$cos(\theta_o) - cos(\alpha + \alpha_o) \leq 0$$

requires

$$\theta_o \geq \alpha + \alpha_o$$

or

$$\alpha \leq \theta_o - \alpha_o$$

Since α is defined as positive,

$$\alpha_o \leq \theta_o$$

and if $\alpha_o \leq \theta_o$ and α is in the range [0^o, $\alpha \leq \theta_o - \alpha_o$], β can be any value in range [0^o, 360^o].

Next consider the case when

$$cos(\theta_o) - cos(\alpha + \alpha_o) > 0$$

We get

$$\theta_o < \alpha + \alpha_o \tag{3.10}$$

Since α is defined as positive, from Equation (3.10),

$$\theta_o - \alpha_o < 0$$

$$\alpha_o > \theta_o$$

With reference to Figure (3.5) (d),

$$\alpha_{min} = \alpha_o - \theta_o > 0$$

Since both α_o and α are greater than 0^o, from the right hand side of Equation (3.6) we write,

$$[cos(\beta - \beta_o) + 1]sin\alpha sin\alpha_o \geq cos(\theta_o) - cos(\alpha + \alpha_o)$$

Since both α_o and α are greater than 0^o,

$$sin\alpha sin\alpha_o > 0$$

$$cos(\beta - \beta_o) \geq \frac{cos(\theta_o) - cos(\alpha + \alpha_o)}{sin\alpha sin\alpha_o} - 1$$

$$cos(\beta - \beta_o) \geq \frac{cos\theta_o - cos\alpha_o cos\alpha}{sin\alpha sin\alpha_o}$$

$$|\beta - \beta_o| \leq arccos(\frac{cos\theta_o - cos\alpha_o cos\alpha}{sin\alpha sin\alpha_o}) \tag{3.11}$$

where

$$\left|\frac{cos\theta_o - cos\alpha_o cos\alpha}{sin\alpha sin\alpha_o}\right| \leq 1 \tag{3.12}$$

Equation (3.12) can be rewritten as,

$$-sin\alpha sin\alpha_o \leq cos\theta_o - cos\alpha_o cos\alpha \leq sin\alpha sin\alpha_o$$

$$cos\alpha_o cos\alpha - sin\alpha sin\alpha_o \leq cos\theta_o \leq cos\alpha_o cos\alpha + sin\alpha sin\alpha_o$$

$$cos(\alpha + \alpha_o) \leq cos\theta_o \leq cos(\alpha - \alpha_o) \tag{3.13}$$

Since angle θ_o is the maximum α deviation from α_o,

$$\theta_o \geq |\alpha - \alpha_o|$$

and the right hand side of Equation (3.13) is always satisfied. The left hand side of Equation (3.13) requires

$$\alpha + \alpha_o \geq \theta_o$$

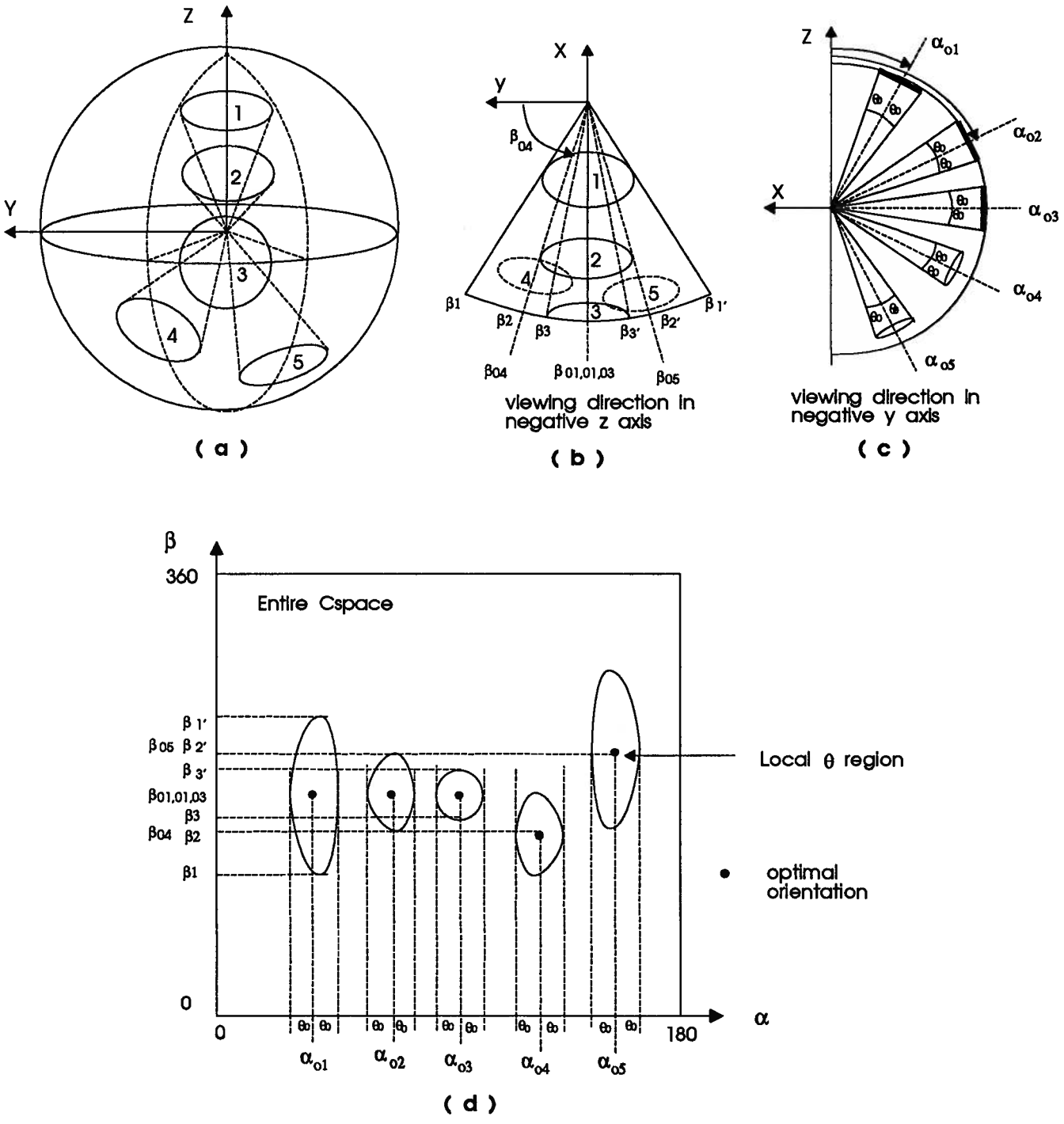

Figure 3.7: Entire C-space and *local θ region*

which is satisfied by the condition (3.10), and condition (3.12) is always true.

Summarizing this discussion,

1. If $\alpha_o > \theta_o$, Equation (3.11) represents the relationships between $\alpha, \beta, \alpha_o, \beta_o$ and θ_o. Each α angle is associated with two β angles, which correspond to two β boundaries corresponding to the α angle on the *local θ region.* The *local θ region* is obtained by calculate the β range for every α value as α increases from (α_o - θ_o) to (α_o + θ_o) with step size 1^o.

2. If $\alpha_o \leq \theta_o$, the orientation where α equals 0^o is included in the *local θ region.*

   (a) If α is in the range [0, α_o-θ_o], the β is the full range [0, 360].

   (b) If α is in the range [α_o-θ_o, α_o+θ_o], the β range is calculated with Equation (3.11).

In Equation (3.11), the β range of the *local θ region* depends on α_o, β_o and θ. The geometric representation of a *local θ region* is given in Figure 3.7. Figure 3.7 (a) illustrates the *local θ regions* for different α_o and β_o angles. Figure 3.7 (b) illustrates that the β_o angle sequence is: $\beta_{o4} < \beta_{o1}, \beta_{o2}, \beta_{o3} < \beta_{o5}$, and different β_o angles correspond to different β range in *local θ region.* Figure 3.7 (c) illustrates that the α_o angle is in the order of: $\alpha_{o1} < \alpha_{o2} < \alpha_{o3} < \alpha_{o4} < \alpha_{o5}$. The α range is $2\theta_o$ and does not vary with α_o for different *local θ regions.* With reference to Figure 3.7 (d), the closer the α_o to 90^o, the smaller the β range. A smaller α_o angle, however, does not change the α range. Different α_o angles, therefore, produce different position, size and shape of *local θ region*, while different β_o angles only change the position of the *local θ regions.*

If a *local θ region* crosses the boundary of the *Entire C-space* region, the *local θ region* has different shape from those discussed above. In Figure 3.8 (a) cone a is divided by the plane where β is equal to 0^o, and the corresponding θ *region* is divided into two parts

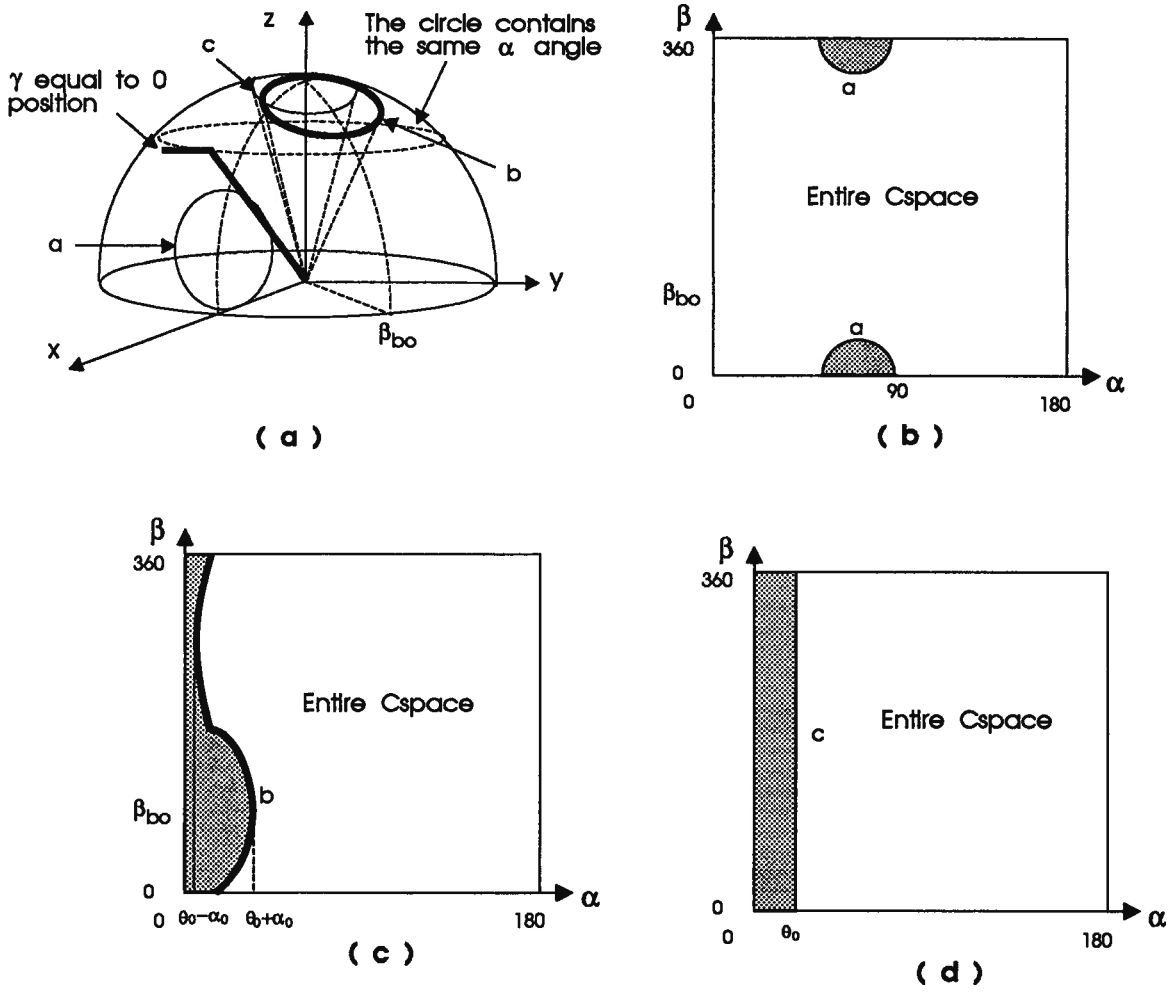

Figure 3.8: The *local θ region* of *Local C-space* in the boundary of Entire C-space

denoted by region a in Figure 3.8 (b). Cone b includes the orientation with both α and β equal to 0^o. In this case, α_o is less than θ_o as discussed above. With reference to Figure 3.8 (c), The corresponding *local θ region* consists of two parts: (1) the α range is [0, α_o-θ_o] and β range is [0,360], and (2) the α range is [α_o-θ_o, α_o+θ_o] and the β range is calculated with Equation (3.11). The special case of α_o equal to 0^o is illustrated in cone c, and the corresponding *local θ region* is rectangular, where α is in the range [0, θ_o] and β is in the range [0, 360], as illustrated in Figure 3.8 (d).

Since the 0^o position of the γ angle is defined at the negative Y axis when both α and β are at 0^o position, refer to Section 3.1, the 0^o position of γ angle always represents the tangent of a circle on the sphere. The circle is determined by the α angle, as shown in Figure 3.8 (a).

Chapter 4

Description of the Midpoint Algorithm

4.1 Overview of the Midpoint Algorithm

The trajectory for a robot tool tracing a pre-defined curve is planned in three stages. First, an extensive search along path segments between successive points is carried out. Each path segment contains two parts: the track of the tool tip and the acceptable orientation range of the tool head. Robot constraints (joint limits), motion constraints (simultaneous translation and rotation which provides for smooth motion), collision avoidance, and the allowable distance between the tool tip and the curve are also considered within this search. Second, the path segments between successive points are linked together using an A^* search to generate a continuous path along the curve. Third, tool configurations along the path are selected using a second A^* search in order to build the optimal trajectory which avoids robot configuration changes. Acceptable process quality is maintained along this trajectory.

4.2 The Description of the Midpoint Algorithm

Given a curve with sufficient curve points to represent the curve shape, the trajectory is planned as follows,

STEP 1.0 Reconstruct the curve using a Catmull-Rom spline in order to get curve point information at any curve position. Details of this step are presented in Section 4.2.1.

STEP 2.0 Use the *Point Check* procedure to identify collision free regions at the first curve point. Store the free regions in a cost-ordered linked list (called *path list*), and use each free region as the beginning of a potential path along the curve. Details of this step are discussed in Section 4.2.2, and the cost function is described in Section 4.2.7.

STEP 3.0 Extend the lowest cost path in the *path list* along the curve by exploring the path segment between successive points in the following steps,

STEP 3.1 *Curvature Check*: Check if successive points are close enough to represent the curve segment. Details of this step are discussed in Section 4.2.3.

STEP 3.2 *Overlap Check*: Check the overlap between the *Local C-spaces* at successive points. This ensures that the trajectory remains within the allowable deviation in tool orientation. Details of this step are discussed in Section 4.2.4. If this check fails, add a midpoint between the points and return to STEP 3.1.

STEP 3.3 *Translation and Rotation Check*: Ensure that the tool can translate and rotate along the path segment without collision. The details of this step are discussed in Section 4.2.5. If this check fails and the step size between current two points is larger than the *minimum step size*[1], add a midpoint and return to STEP 3.1. Otherwise, the failure in finding a path is reported to user and trajectory planning stops.

STEP 3.4 *Inverse Kinematics Check*: Check the validity[2] of robot inverse kinematic solutions corresponding to the robot tool configurations at the point. The details of this step are discussed in Section 4.2.6. If this check fails, delete the current path from the *path list*. If the *path list* is empty, the trajectory

[1]The *minimum step size* is predetermined in order to avoid the infinite exploration of a path segment.
[2]A valid inverse kinematic solution does not violate robot joint limits.

planning stops. Otherwise, resume search at STEP 3.0.

STEP 3.5 Repeat STEP 3.0 until the final curve point is included in a path in the *path list* or no possible path remains in the *path list.*

STEP 4.0 The resulting path contains different α, β and γ ranges between successive curve points. Each combination of α, β and γ is the tool configuration that may be included in the final trajectory. An A^* search is used to select the configurations in the path in order to obtain the optimal trajectory along the curve. The details of this step are discussed in Section 4.2.8.

A fundamental component of this algorithm is the addition of midpoints between successive curve points as the search progresses. The reason for adding midpoints is that if successive points are close enough, the straight line motion of the tool tip between these two points will match the curve segment within a given tolerance. In this case, the optimal orientations at the points will be "close", and some tool configurations at successive points will share collision-free space. As a result, there will be translation and rotation collision-free path segments between successive points. The concept of "close" depends on the curvature of the curve and environment in the vicinity of the points. The more complex the curve or the environment, the closer the successive points. For example, in Figure 4.1 (a), the curve, the optimal orientation and the environment do not change significantly, and the points p_i and p_{i+1} are considered as close enough for the process. In Figure 4.1 (b), the environment and the optimal orientations do not change significantly but the curve changes sharply. In Figure 4.1 (c), the environment does not change but the optimal orientations and the curve changes sharply between points p_i and p_{i+1}. In Figure 4.1 (d), the curve and the optimal orientations do not change much but the environment changes sharply. In case (b), (c) and (d), the two curve points p_i and p_{i+1} are not close enough for the motion planning process. A smaller step size is required

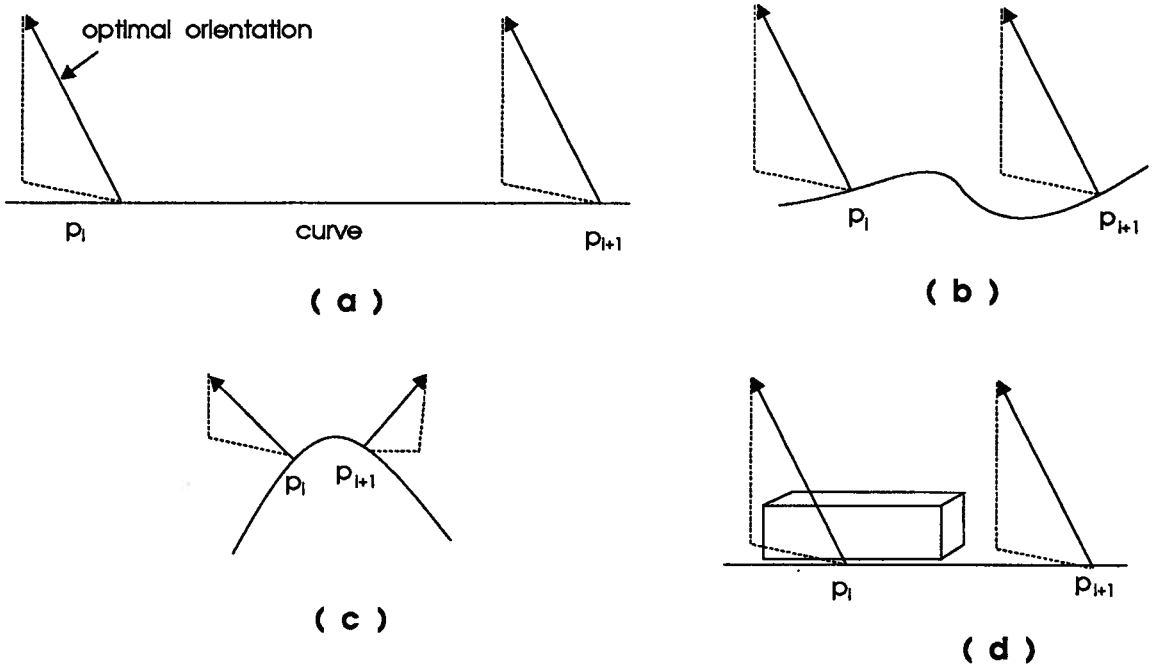

Figure 4.1: The relation between step size and the condition of environment and curve

to explore the path in Figure 4.1 (b), (c) and (d). A larger step size, however, tends to speed up motion planning in Figure 4.1 (a). The insertion of midpoints between p_i and p_{i+1} controls the step size.

A midpoint is added in the way that the location of the midpoint is on the curve midway between two curve points. The optimal orientation at the midpoint is determined as described in Section 1.1 for determining the optimal orientation at the original curve points.

4.2.1 Reconstruct the Curve by Catmull-Rom spline

A Catmull-Rom spline [Catm 74] is used to reconstruct a curve based on the sample points which lie on the desired curve. The sample points are used as the control vertices, and the Catmull-Rom splines interpolate the control vertices. These splines share many properties with B-splines, such as global smoothness and local control. One member of

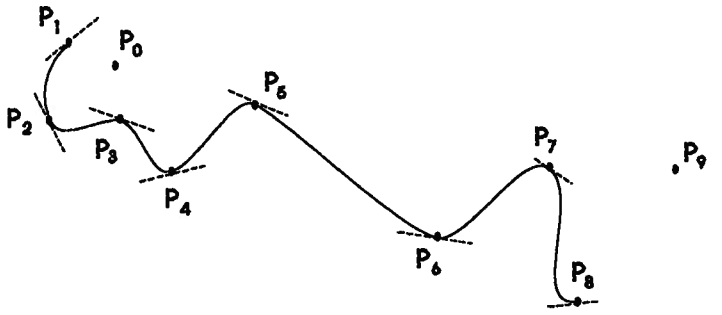

Figure 4.2: A Catmull-Rom spline

Catmull-Rom spline family is able to interpolate the points P_1 to P_{m-1} from the sequence of points P_0 to P_m. The tangent vector at point P_i is parallel to the line connecting point P_{i-1} and P_{i+1}, as shown in Figure 4.2.

Catmull-Rom spline uses parametric representation of curves. Here a curve is approximated by a piece-wise polynomial curve. Each segment $Q^i(t)$ of the overall curve is given by three functions, $x = x(t), y = y(t), z = z(t)$, which are cubic polynomials in the parameter t, $(0 < t < 1)$. Given points p_i, p_{i+1} and new point p_{new} between p_i and p_{i+1}, t is the ratio of the distance between p_i and p_{new} and the distance between p_i and p_{i+1}. Designating M_{CR} as the Catmull-Rom basis matrix, G^i_{Bs} as the geometry matrix, and T as the matrix of parameter t. With $T = [t^3 t^2 t^1 1]$, the representation is

$$Q^i(t) = T \cdot M_{CR} \cdot G^i_{Bs} = \left[\begin{array}{cccc} t^3 t^2 t^1 1 \end{array}\right] \frac{1}{2} \left[\begin{array}{cccc} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{array}\right] \left[\begin{array}{c} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{array}\right] \tag{4.1}$$

Where $P_{i-3}, P_{i-2}, P_{i-1}$ and P_i can be any of x, y, or z coordinates of the point $p_{i-3}, p_{i-2}, p_{i-1}$ and p_i, and $Q^i(t)$ is the corresponding function for x, y or z along the curve.

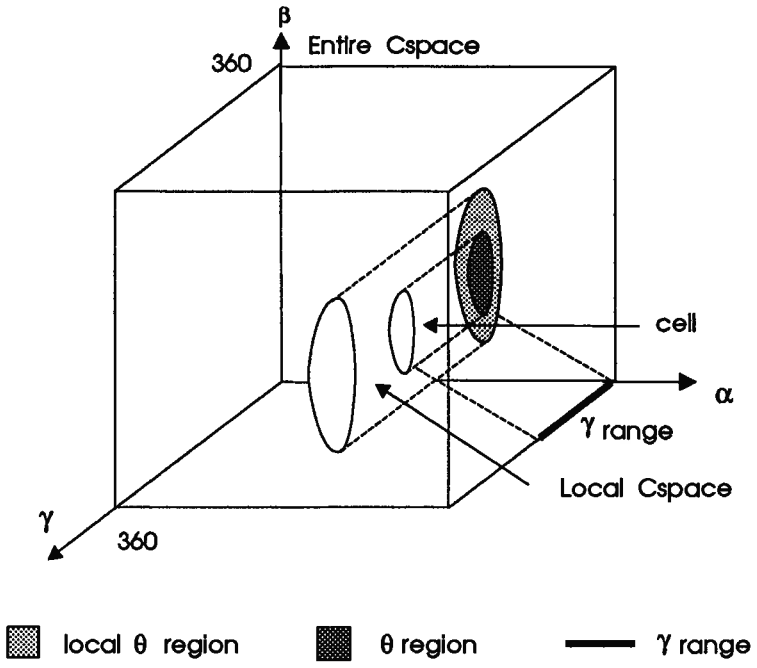

Figure 4.3: The definitions of *cell, θ region* and *γ range*

Using a Catmull-Rom Spline, the first and last points are not included in the curve. The problem can be solved as follows,

1. If the curve is closed, the last and first points are used again as the first and last points.

2. If the curve is open, the first and last points are used twice.

4.2.2 Point Check for the First Point

Point Check

The goal of the *Point Check* is to explore the collision-free space around a curve point. Converting the problem into the configuration space, this check becomes to search for *free cells* in the *Local C-space*. With reference to Figure 4.3, a *cell* is defined as the space in *Local C-space* which projects as the *θ region* on the $\alpha - \beta$ plane. The *cell* also projects

as γ *range* on the γ axis. If the corresponding swept volume of a *cell* is placed at the curve point corresponding to the *Local C-space*, and it does not collide with the environment, the *cell* is called a *free cell.* The swept volumes of the robot tool are the bridges between the configuration space and the work space. Each curve point is associated with a *Local C-space* and each *cell* in the *Local C-space* is associated with a swept volume of the robot tool. The shape of the swept volume is defined by the α, β and γ ranges of the *cell.* Although the *Local C-spaces* with different α_o and β_o angles have different shapes and sizes as discussed in Chapter 3, the shapes of the corresponding swept volumes are the same. As a result, only one set of swept volumes and its subdivision parts are required, and this feature makes the Midpoint Algorithm practical for implementation on existing PC hardware.

Two kinds of swept volumes are used in the check: the θ *swept volume* and the γ *swept volume* which correspond to the θ *region* and the γ *range* respectively, i.e. the θ *swept volume* and the γ *swept volume* represent the space occupied by the tool head and the tool handle respectively. In Figure 4.4 (a), placing the tool with the critical tool orientation parallel to the Z axis and rotating the tool $\pm\theta$ angle ($0 < \theta < \theta_o$) about X axis generate the region swept by the tool. The regions swept by the tool head and the tool handle are called the θ *swept region* and the γ *swept region* respectively. The θ *swept volume* and the γ *swept volume* are generated by rotating the θ *swept region* and the γ *swept region* 180^o about Z axis as shown in Figure 4.4 (b). Figure 4.4 (c) and (d) provide cross-sectional views of the θ *swept volume* and the γ *swept volume.* A set of θ *swept volumes* with different θ angles are generated as shown in Figure 4.5 (a). For each θ *swept volume*, there is a set of corresponding γ *swept volumes* associated with. With reference to Figure 4.5 (b), the set of γ *swept volumes* are generated by dividing the full range γ *swept volume* with two planes which are the tangent planes of the θ *swept volume.* The angle ψ between these planes is equal to $\frac{180^o}{n}$, where $n = 1, 2, ...$, and a binary tree data

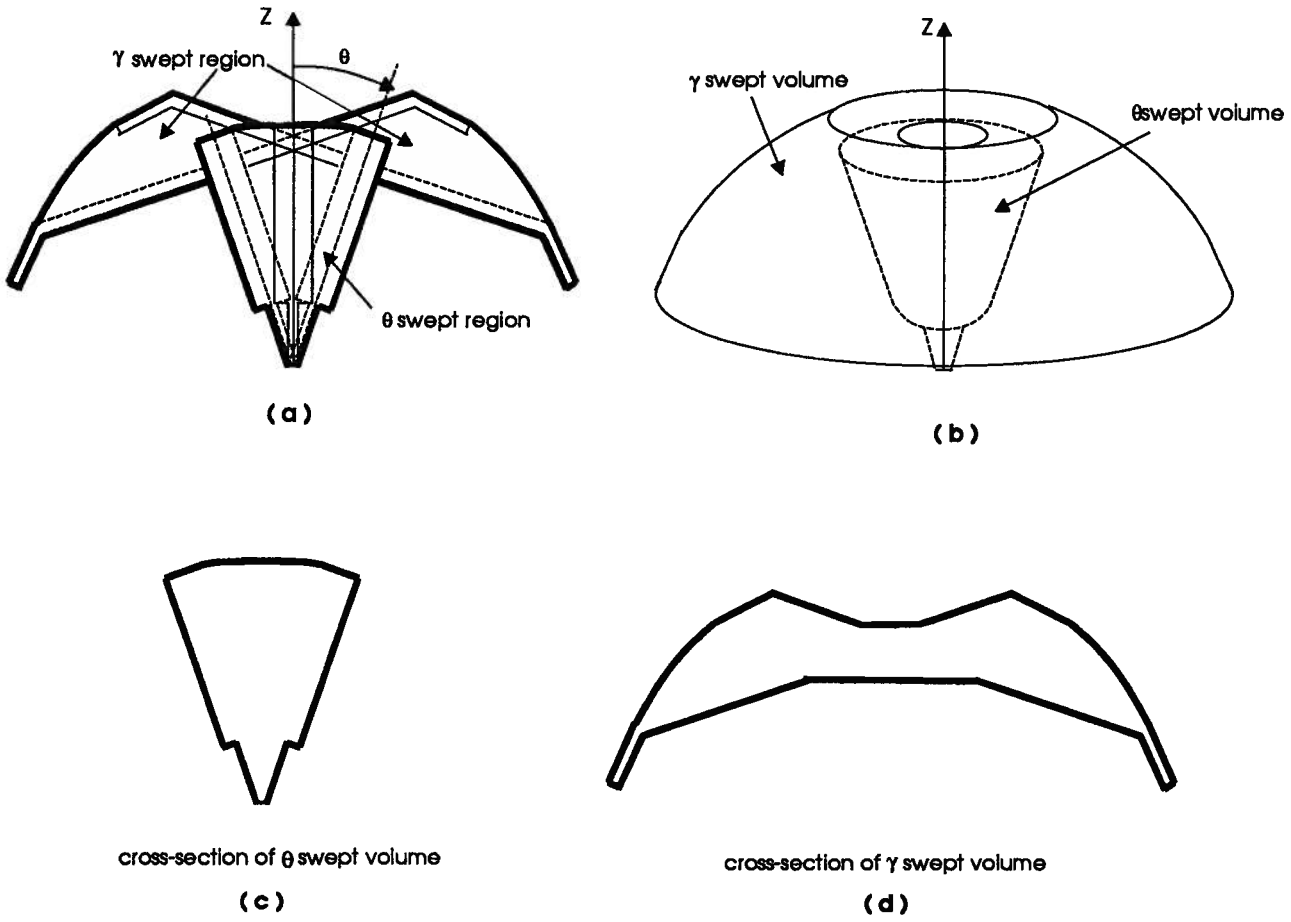

Figure 4.4: The swept volumes used in the *Point Check*. (a) The θ *swept region* and the γ *swept region*. (b) The θ *swept volume* and the γ *swept volume*. (c) The cross-section of θ *swept volume*. (d) The cross-section of γ *swept volume*.

structure is generated. Note that the full range γ *swept volume* corresponds to $n = 0$. The cases of $n = 1$ and $n = 2$ is shown in Figure 4.5 (c). If the θ *swept volume* contains Z axis, i.e. $\alpha_o < \theta_o$, the corresponding γ *swept volume* and its decomposed swept volumes are always the full range γ *swept volume*. The combination of a θ *swept volume* and a γ *swept volume* is shown in Figure 4.5 (d).

At a curve point, the corresponding θ *region* of a θ swept volume at a curve point is computed based on the optimal orientation at the point and the θ angle of the swept volume. The details of this computation are discussed in Chapter 3.

The *Point Check* searches the *free cell* in a *Local C-space* in two steps: (1) shrink the θ *region* in order to search for the collision-free region (*free* θ *region*), and (2) search the binary tree for collision-free ranges in γ *range* for collision-free ranges (*free* γ *ranges*). The combination of a *free* θ *region* and a *free* γ *range* corresponds to a *free cell*.

Search the First Curve Point

The *Point Check* is applied at the first curve point in order to search for a *free cell* in the corresponding *Local C-space*. Each *free cell* is the beginning of a potential path along the curve. The *free cells* are stored in the *path list* and sorted by the *PathCost* of the *free cell*. The details of the *PathCost* calculation are discussed in Section 4.2.7.

If no *free cell* is found, trajectory planning fails since the robot tool cannot be placed at the first curve point without collision. A high resolution representation of the *Local C-space* is required to find small *free cells*. A high resolution *Local C-space*, however, requires a huge tree structure, which takes much time to store and search the data, and can lead to implementation problems. This detail is discussed in Chapter 5.

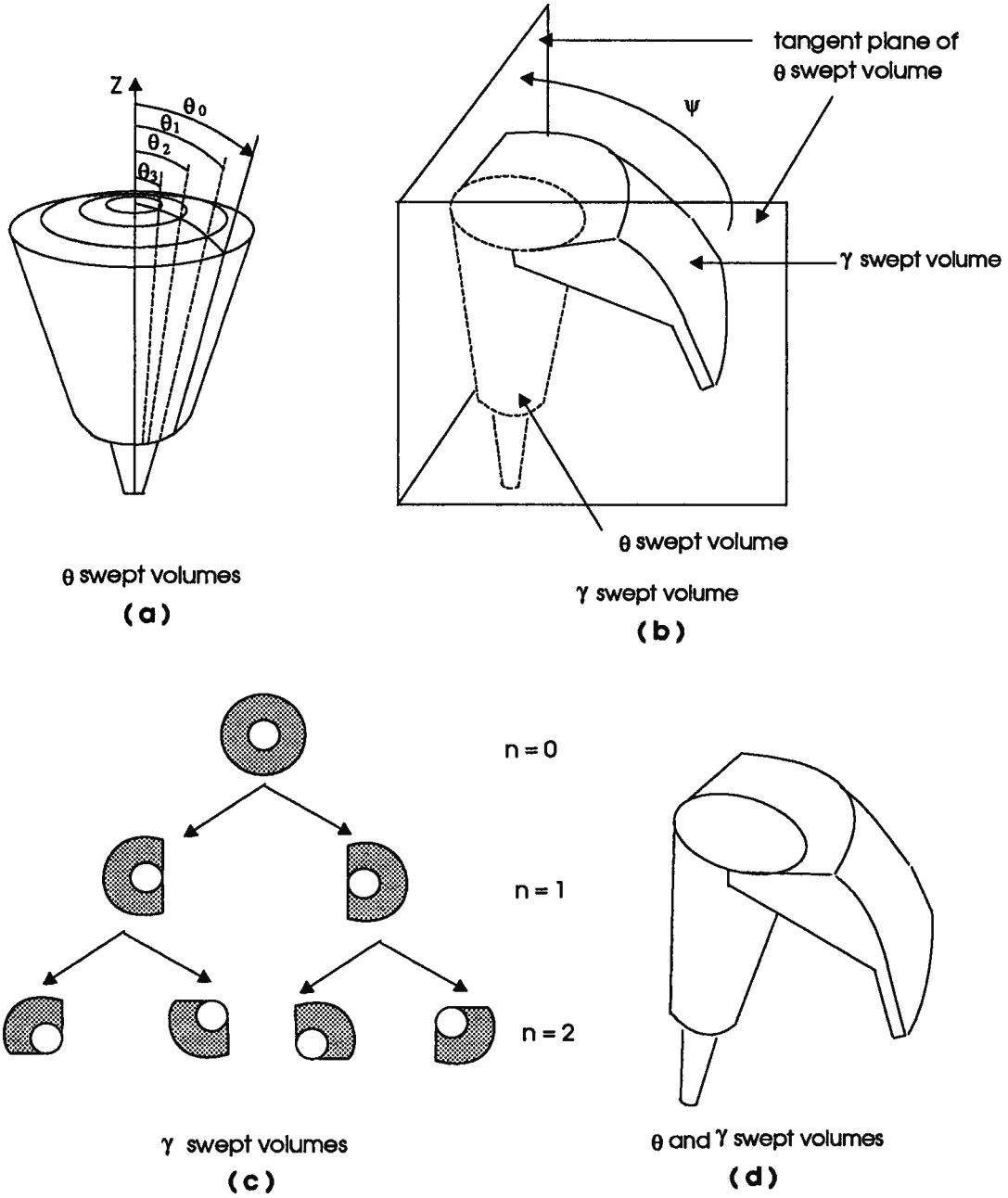

Figure 4.5: Swept volumes used in *Point Check* (a) The division of θ *swept volume* (b) The division of γ *swept volume* (c) The binary tree structure of γ *swept volume* (d) The combination of a θ *swept volume* and a γ *swept volume*

4.2.3 Curvature Check

Since a curve is represented by a set of discrete curve points, the motion of the tool along the curve is decomposed into the motion between successive curve points. Using a fixed step size between the curve points may not be practical depending on the complexity of the environment in the vicinity of the curve and the degree of curvature. For instance, using a very small step size along the curve may waste search time along the curve segments where the curvature is low and the surrounding environment is simple. Using a step size which is large, however, may result in the tool tip deviating from the required curve along curve segments where the curvature is high. This will result in poor process quality, or damage to the tool. It is difficult to predetermine a suitable fixed step size, and a mechanism for automatically adjusting the step size based on the nature of the curve and the environment is necessary.

One of the advantages of the Midpoint Algorithm is that the step size is varied automatically along the curve. If the curvature along a segment is low, a large step size is used to speed up the trajectory planning along the segment. Otherwise, a small step size is used to ensure acceptable translation of the tool along the curve. In general, the more complex the curve, the smaller the step size.

Adjustment of the step size based on curvature is discussed in this section, while adjustment based on the complexity of the environment is discussed in Section 4.2.5. The *Curvature Check* not only varies the step size along different curve segments, it also searches a polyline which matches the shape of curve within a given tolerance and is used as a reference for the tool tip. The polyline is generated by detecting if the straight line connecting successive points can represent the curve segment within the tolerance. Some points may be added or deleted based on the curvature of a curve,

1. If the curvature of a curve segment between successive points is high, the curve

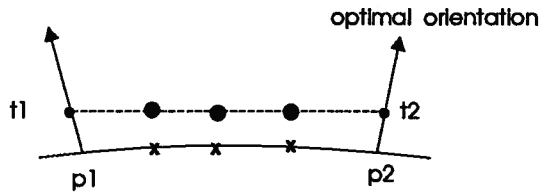

Figure 4.6: Intermediate points selected from curve segment and line segment

segment cannot be represented by the straight line between the points. Further midpoints are added until the curvature of the new curve segments are within an acceptable tolerance of the original curve.

2. If there are several points on a flat section of the curve, some points are deleted in order to speed up the search process.

The tolerance discussed above corresponds to the two parameters, d_{max} and d_{min}, the maximum and minimum distances between the tool tip and the curve. Both of these distances are specified by the application. In most of the cases, d_{min} is greater than 0 to ensure that the tool does not collide with the workpiece. For each curve point, a track point is generated that is positioned a distance d_{max} in the direction of optimal tool orientation. These track points correspond to the reference position for the tool tip. With reference to Figure 4.6, a number of intermediate points, called *intermediate curve points*, are selected from the curve segment between successive curve points p_1 and p_2, and the same number of points, called *intermediate line points*, are selected from the line segment connecting the corresponding track points t_1 and t_2. The number is called the *intermediate number*. The optimal orientations at the *intermediate curve points* are specified as described in Section 1.1 for the original curve points. The following two tests examine if the line segment connecting successive track points is an acceptable

representation of the required tool path.

1. *Vector Test*: for each *intermediate line point*, if the angle between the optimal orientation at the corresponding *intermediate curve point* and the vector from this *intermediate line point* to the corresponding *intermediate curve point* is greater than 90°, the *Vector Test* succeeds.

2. *Distance Test*: for each *intermediate line point*, if the distance between the *intermediate line point* and its corresponding *intermediate curve point* are greater than d_{min} and less than d_{max}, the *Distance Test* succeeds.

If one of the above tests fails, midpoints and corresponding track points must be added until both tests succeed. The addition of midpoints is illustrated in Figure 4.7 and summarized as follows,

1. Since the *Vector Test* fails between *intermediate line point* l_1 and *intermediate curve point* c_1 in Figure 4.7 (a), midpoint m_1 is added between curve points p_1 and p_2 in Figure 4.7 (b).

2. Since the *Distance Test* fails between *intermediate line point* l_2 and *intermediate curve point* c_2 in Figure 4.7 (c), midpoint m_2 is added between curve points p_1 and m_1 as illustrated in Figure 4.7 (d)

3. In Figure 4.7 (d), no midpoint is required between curve points p_1 and m_2, and between m_2 and m_1 since both the *Vector Test* and the *Distance Test* succeed in these curve segments.

4. Since the *Vector Test* fails between *intermediate line point* l_3 and *intermediate curve point* c_3 in Figure 4.7 (e), midpoint m_3 is added between curve points m_1 and p_2 In Figure 4.7 (f)

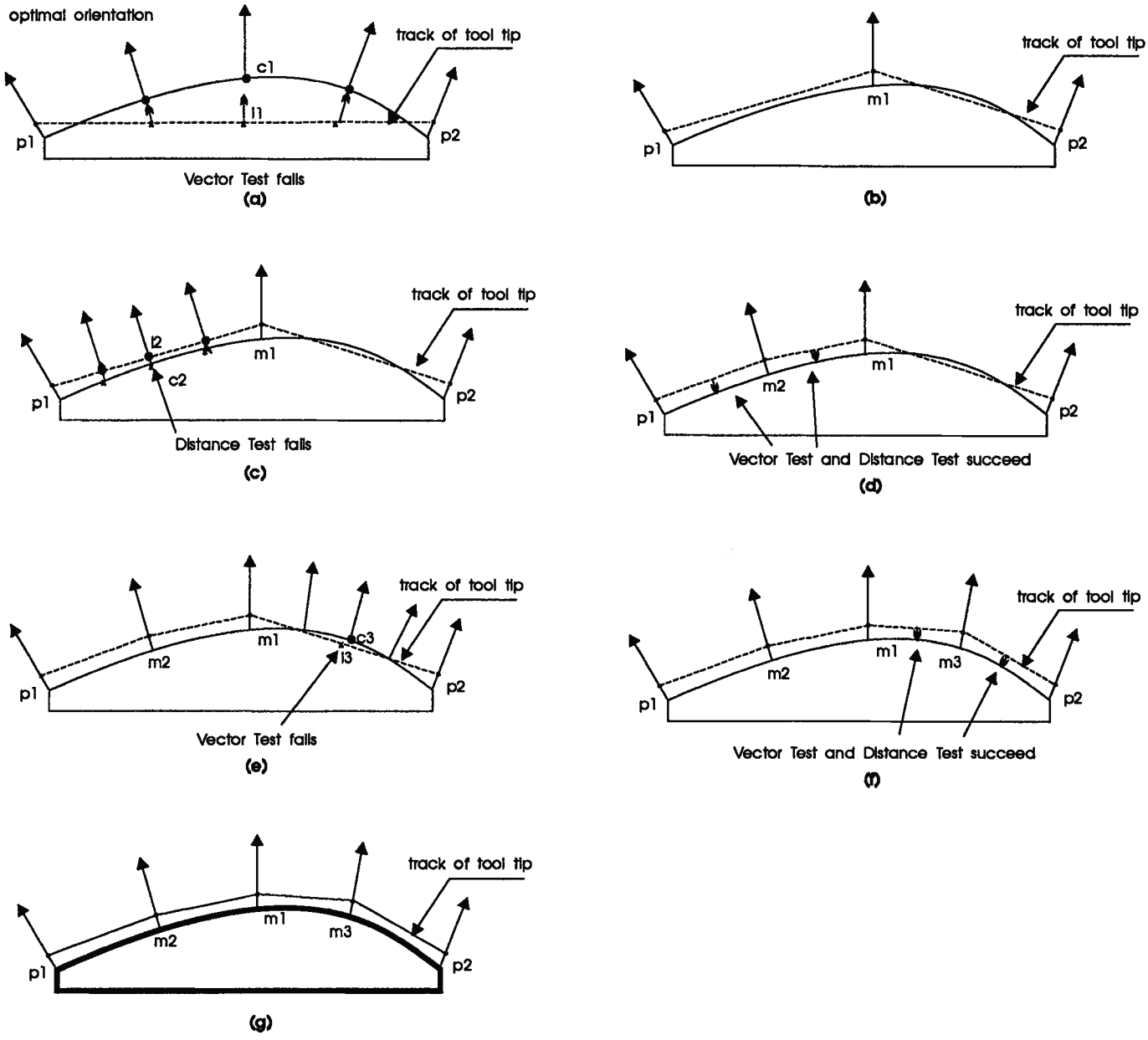

Figure 4.7: Adding midpoints using *Curvature Check*

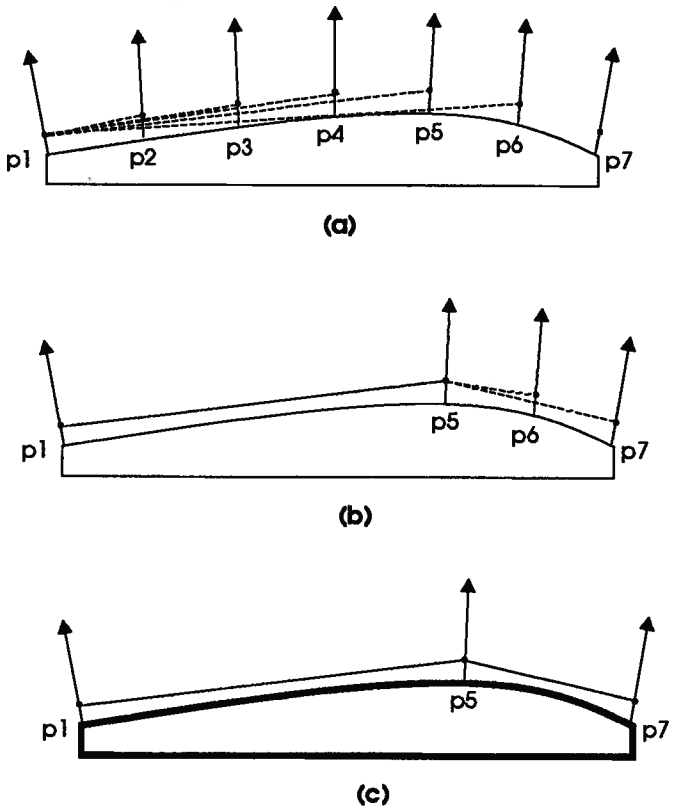

Figure 4.8: Deleting curve points using *Curvature Check*

5. In Figure 4.7 (f), no midpoint is required between curve points m_1 and m_3 or between m_3 and p_2 since both the *Vector Test* and the *Distance Test* succeed in these curve segments.

6. The final track of the tool tip redefined by the *Curvature Check* is shown in Figure 4.7 (g).

The *Curvature Check* is also used to delete curve points which are not necessary in the curve segment where the curvature is low. If the *Curvature Check* between p_i and p_{i+1} succeeds, the curvature between p_i and p_{i+2} is checked. The number of *intermediate line points* and *intermediate curve points* selected must have the same density as the points specified by the *intermediate number*. For instance, if the *intermediate number* is

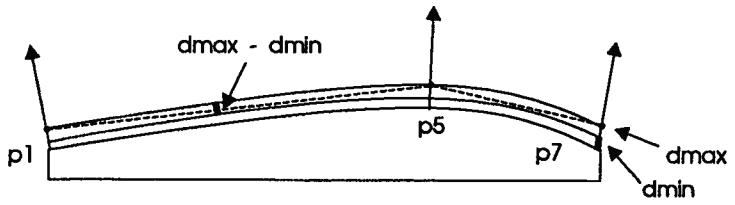

Figure 4.9: The relationship between track and constraints: d_{max} and d_{min}

3, three *intermediate curve points* and three *intermediate line points* are selected between p_i and p_{i+1} and their corresponding track points, and six *intermediate curve points* and six *intermediate line points* are selected between p_i and p_{i+2} and their corresponding track points, and so on. If the *Curvature Check* between p_i and p_{i+2} fails, the *Curvature Check* between p_i and p_{i+1} finishes, and no curve point is deleted from the curve. Otherwise, the curvature between p_i and p_{i+3} is checked, and so on, until the *Curvature Check* between p_i and p_{i+k} fails. All the curve points between p_i and p_{i+k-1} are then deleted from the curve and the path segment between p_i and p_{i+k-1} is planned directly. The deletion of curve points is illustrated in Figure 4.8 and summarized as follows,

1. In Figure 4.8 (a), since the *Vector Test* and the *Distance Test* succeed between p_1 and p_2, between p_1 and p_3, between p_1 and p_4, between p_1 and p_5, but fail between p_1 and p_6, the curve points p_2, p_3 and p_4 and the corresponding track points are deleted.

2. In Figure 4.8 (b), since the *Vector Test* and the *Distance Test* succeed between p_5 and p_6, between p_5 and p_7, and p_7 is the last curve point, the curve point p_6 and its corresponding track point are deleted.

3. The final track of the tool tip searched by *Curvature Check* is shown in Figure 4.8 (c).

The precision of the *Curvature Check* is determined by the parameter d_{max}, d_{min} and

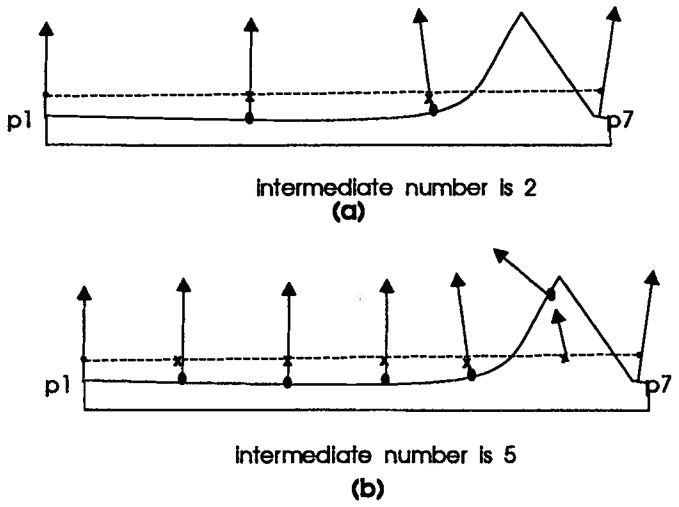

Figure 4.10: The *intermediate number* and the precision of *Curvature Check*

intermediate number. Since the upper and lower boundaries of the track of the tool tip are specified by d_{max} and d_{min} respectively as shown in Figure 4.9, a small d_{max} and a large d_{min} result in a highly precise track of the tool tip. This will slow down the path planning, however, by adding more midpoints and searching more path segments between successive track points. Since the curvature change may not be detected by a small *intermediate number*, as shown in Figure 4.10 (a), and a large *intermediate number* as shown in Figure 4.10 (b) is necessary. Since the distance calculations are computationally inexpensive, a large *intermediate number* (e.g. 10) is recommended.

4.2.4 Overlap Check

Since the γ ranges in the *Local C-spaces* are identical, the overlap discussed here is the overlap between the θ *regions*. The intersection of two θ *regions* is determined through a search for the overlapping α ranges and the β range at every overlapping α value, as illustrated in Figure 4.11 (a). If no overlapping α range or overlapping β range at overlapping α value is found, such as regions 1 and 3, and regions 1 and 2 in Figure

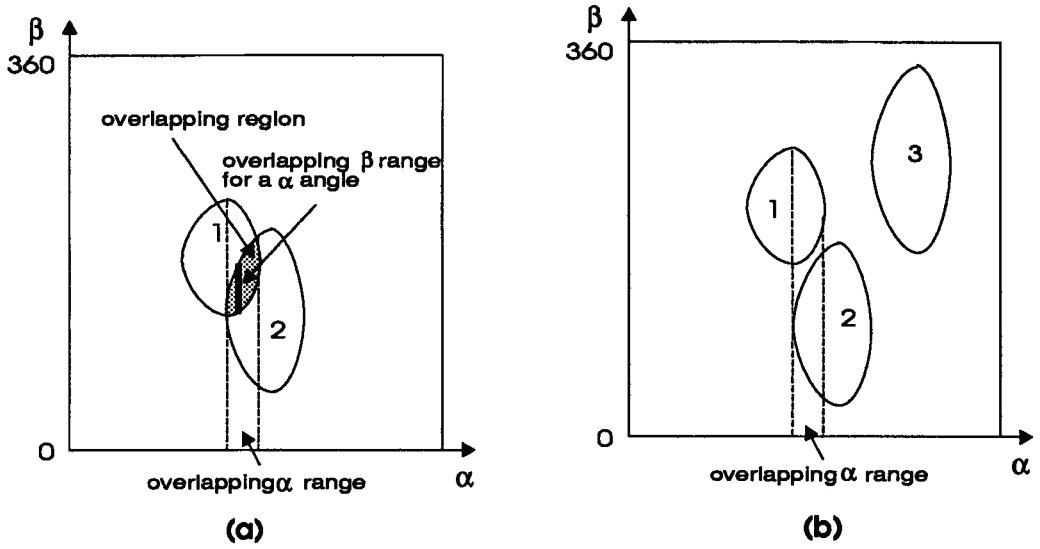

Figure 4.11: The search for overlapping region

4.11 (b), the two θ *regions* do not overlap. The intersection points of the boundary of two θ *regions* are obtained by searching the α value which has the same β value at the boundary in both regions.

If the θ *regions* at successive points overlap, the curve segment between these points can be searched. As illustrated in Figure 4.12 (a) (b), a valid path cannot be found within the allowable orientation range between p_1 and p_2. If the points on a continuous curve are very close, the optimal orientations should also be very close. Since the position of the θ *region* is determined by the optimal orientation, the addition of midpoints eventually makes the θ *regions* at successive points overlap as illustrated in Figure 4.12 (c). If the step size between two points is less than the *minimum step size*, a user defined parameter, and the θ *regions* do not overlap, the curve is assumed to be discontinuous between the two points. In this case, the optimal orientation and the position of the midpoint are defined as the bisection of the orientations and the positions at the two points, and midpoints are added until the θ *regions* overlap. This process, which is illustrated in

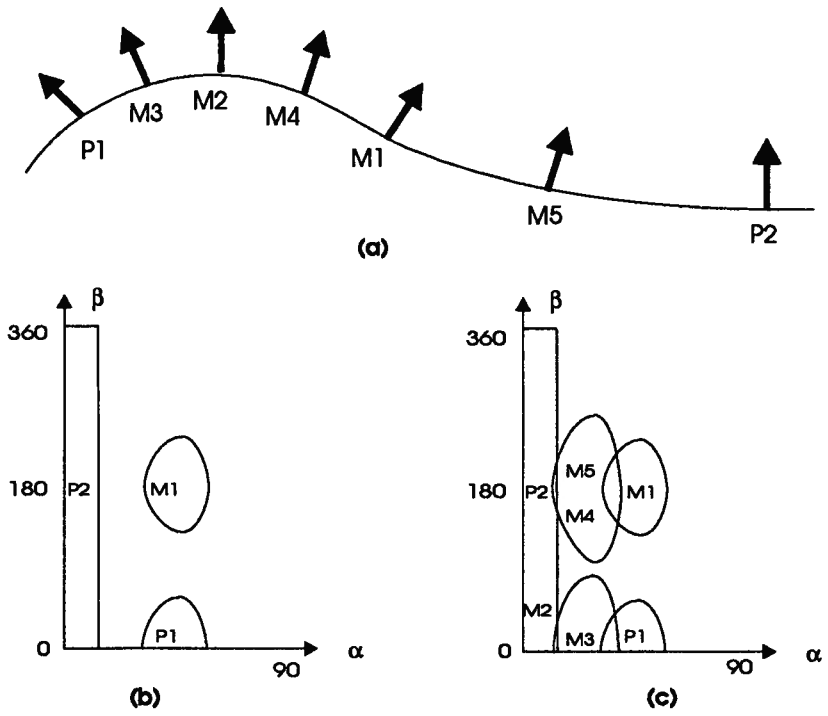

Figure 4.12: The effect of adding midpoint

Figure 4.13[3], improves the robot motion passing a corner or a discontinuous curve point.

The addition of a midpoint may not lead to smooth changing orientations since the optimal orientation at the midpoint of two curve points may not be between the optimal orientations at the curve points. With reference to Figure 4.12 (a) and (c), the optimal orientation at midpoint m_1 is not between the optimal orientations at p_1 and p_2. If m_1 is not added, the robot tool will move between p_1 and p_2 with the orientation interpolated between the optimal orientations at p_1 and p_2, which results in an incorrect trajectory. The addition of m_1 detects the change of curvature and avoids this error. The optimal orientation at midpoint m_2 is between the optimal orientations at p_1 and m_1, and the optimal orientation at midpoint m_3 is between the optimal orientations at p_1 and m_2. The additions of m_2 and m_3 make the optimal orientations at curve points close enough

[3]In Figure 4.13, the curve is on the Y-Z plane and the optimal orientation is defined as the normal of the curve, which maintains an acute angle with the Z axis.

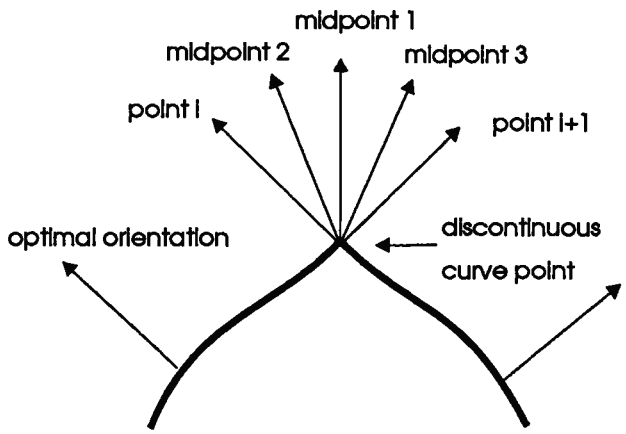

Figure 4.13: Adding midpoints in a discontinuous curve

since their θ *regions* overlap. The additions of m_4 and m_5 are necessary since the θ *regions* of m_1 and m_2, and the θ *regions* of m_1 and p_2 do not overlap respectively[4]. If the curve is not continuous, the addition of midpoints provides for a smooth change of trajectory as illustrated in Figure 4.13.

It is important to consider the relationship between the shape and size of the θ *region* which depends on the α_o angle. In the α_o range [0, 90], the smaller the α_o angle, the larger the β range. In Figure 4.14 (a), orientations 1 and 4 are far apart, while orientations 2 and 3 are close together. Each orientation in the pair have the same α_o angle while β_o is 0^o for orientations 1 and 2 and is 180^o for orientations 3 and 4. The corresponding θ *regions* of 2 and 3 overlap, while the θ *regions* of 1 and 4 do not overlap due to the different β ranges shown in Figure 4.14 (b).

4.2.5 Translation and Rotation Check

Configuration Space Search

[4]The θ *regions* of m_2 and p_2, and the θ *regions* of m_4 and m_5 happen to overlap in this example.

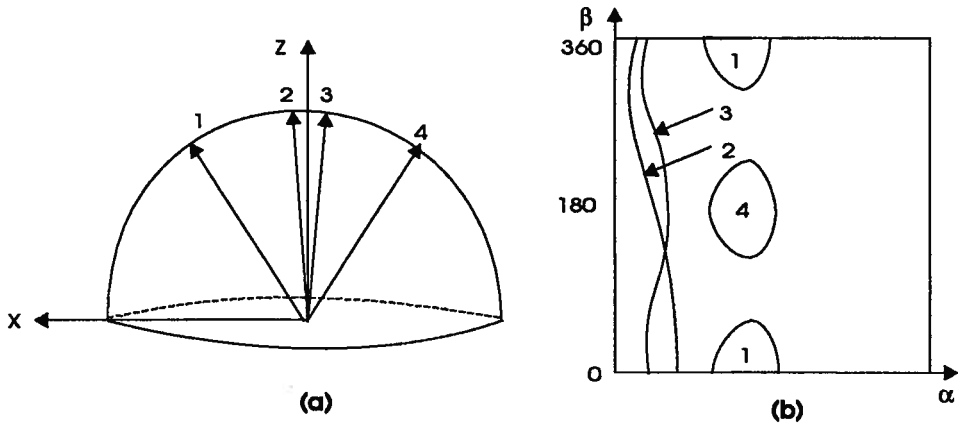

Figure 4.14: The overlap of θ *regions* which have small α angles

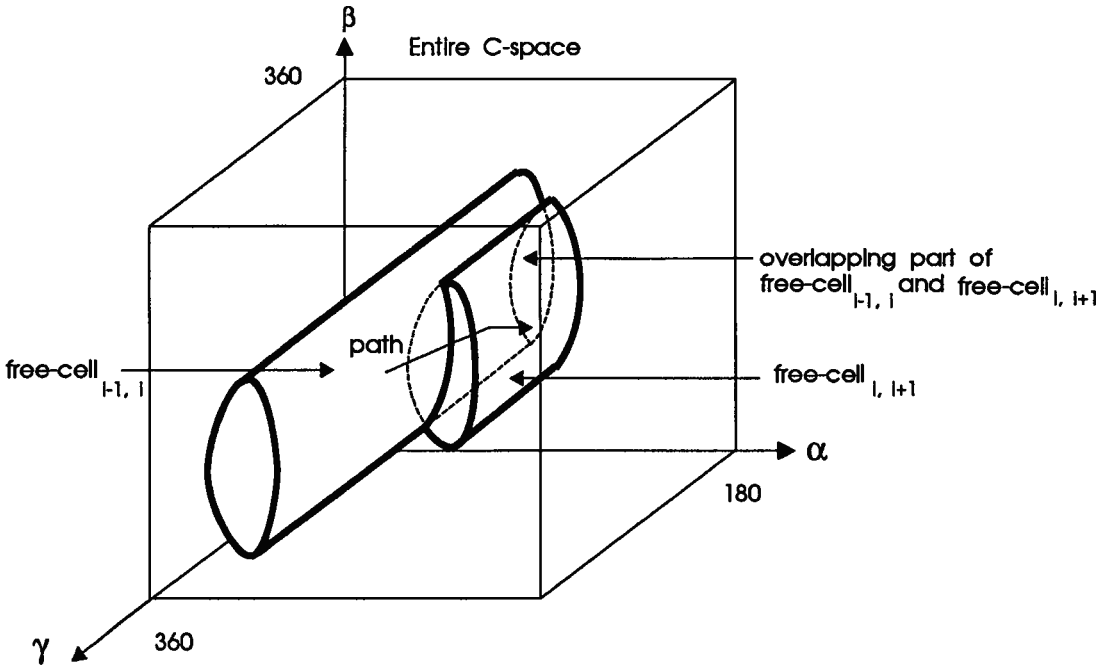

Figure 4.15: The path found though p_{i-1}, p_i and p_{i+1}

This check detects the environment between two track points and searches for a path within the allowable orientation range. The path that is found allows the tool to translate and rotate simultaneously between the points without collision. To meet these requirements, only the overlapping space between two *Local C-spaces* are considered. The overlapping space between the *Local C-spaces* at track points p_i and p_{i+1}[5] is termed *overlap-cell*$_{i,i+1}$[6]. If the robot tool maintains the configurations within part of *overlap-cell*$_{i,i+1}$ and translates between p_i and p_{i+1} without collision, this part of *overlap-cell*$_{i,i+1}$ is called *free-cell*$_{i,i+1}$.If *free-cell*$_{i,i+1}$ overlaps *free-cell*$_{i-1,i}$, *free-cell*$_{i-1,i}$, *free-cell*$_{i,i+1}$ and their overlapping space compose of a safe path through curve points p_{i-1}, p_i and p_{i+1}. This is illustrated in Figure 4.15.

The projections of *overlap-cell*$_{i,i+1}$ and *free-cell*$_{i-1,i}$ on the $\alpha - \beta$ plane are called *overlap-region*$_{i,i+1}$ and *free-region*$_{i-1,i}$ respectively, and the projection of *free-cell*$_{i-1,i}$ on the γ axis is called *free-range*$_{i-1,i}$. The overlap between *free-cell*$_{i-1,i}$ and *free-cell*$_{i,i+1}$ are projected as the overlap between *free-region*$_{i-1,i}$ and *free-region*$_{i,i+1}$ and the overlap between *free-range*$_{i-1,i}$ and *free-range*$_{i,i+1}$. The overlap between *free-range*$_{i-1,i}$ and *free-range*$_{i,i+1}$ is quite straight forward. We mainly discuss the overlap between *free-region*$_{i-1,i}$ and *free-region*$_{i,i+1}$. With reference to Figure 4.16 (a), the *free-region*$_{i-1,i}$ does not overlap with the *overlap-region*$_{i,i+1}$, a midpoint m_i should be added between p_i and p_{i+1}. If *free-region*$_{i-1,i}$ still does not overlap the *overlap-region*$_{i,m_i}$ as shown in Figure 4.16 (b), further midpoints are added between p_i and m_i until *free-region*$_{i-1,i}$ overlaps with the *overlap-region*$_{i,m_k}$ where m_k is the closest midpoint to p_i, as illustrated in Figure 4.16 (c). A small *free-region*$_{i-1,i}$ requires the addition of more midpoints, and a smaller step size is automatically achieved and more careful motion of the robot is obtained as the path

[5]Because of the addition and deletion of curve points, the indices of curve points may not be sequential, and be expressed by integers. In the following text, p_{i+1} represents the next point to p_i and p_{i-1} represents the previous point to p_i.

[6]In the following text, "$_{i,i+1}$" and "$_{i-1,i}$" mean "between p_i and the next point to p_i" and "between p_i and the previous point to p_i" respectively.

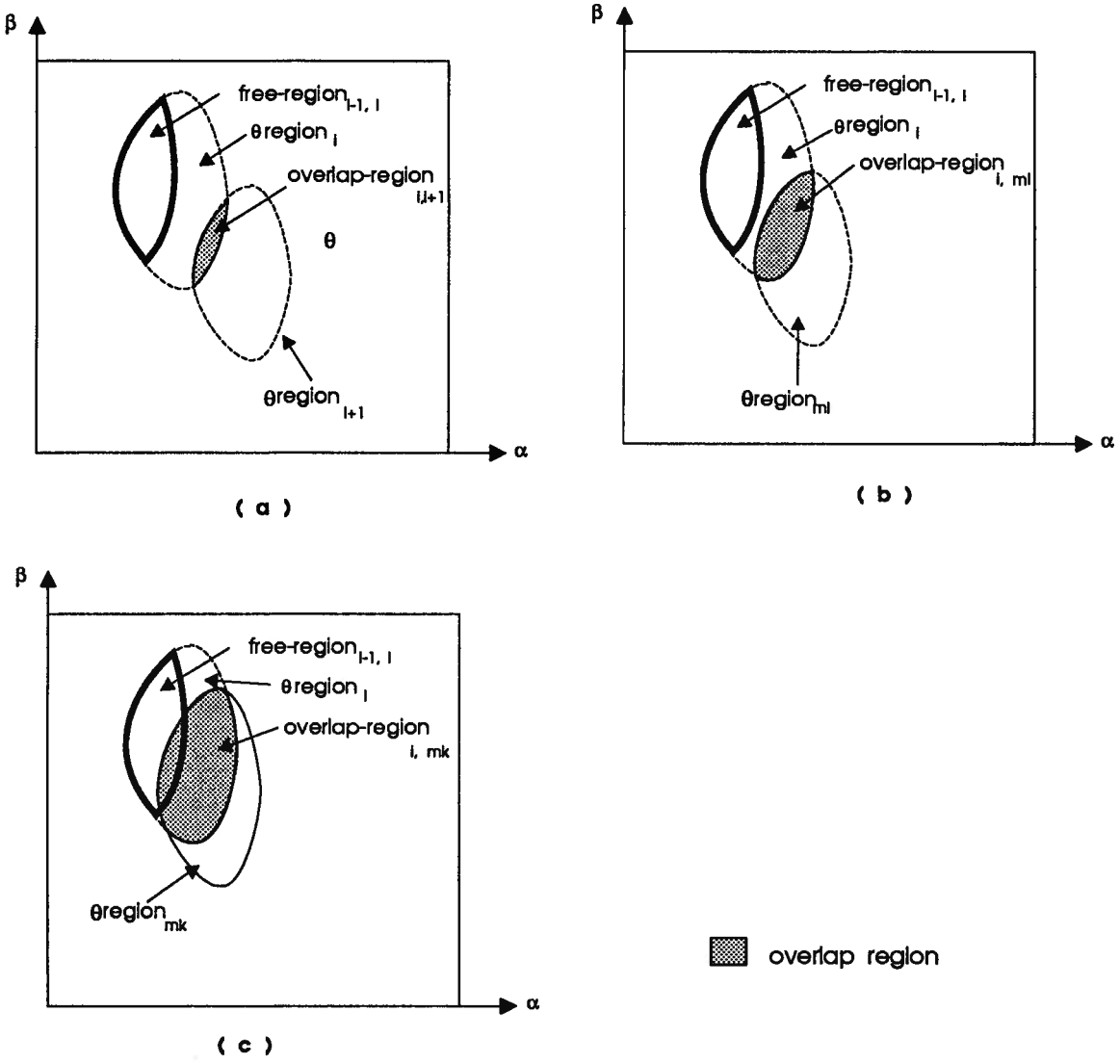

Figure 4.16: The overlap between *free-region*$_{i-1,i}$ and *overlap-region*$_{i,i+1}$

becomes narrow.

After *free-region*$_{i-1,i}$ overlaps with the *overlap-region*$_{i,i+1}$ where p_{i+1} is the next point to p_i [7]. The *free-cell*$_{i,i+1}$ is found by searching for (1) the *free-region*$_{i,i+1}$ in the overlapping region between *free-region*$_{i-1,i}$ and *overlap-region*$_{i,i+1}$, and (2) the *free-range*$_{i,i+1}$ associated with the *free-region*$_{i,i+1}$. The tool head can rotate in *free-region*$_{i,i+1}$, and the tool handle can be placed in any position in *free-range*$_{i,i+1}$ as it moves between p_i and p_{i+1}.

The concepts of *shrink step*, *shrink region* and *minimum shrink region* are illustrated in Figure 4.17. In Figure 4.17 (a), the *overlap-region*$_{i,i+1}$ shifts in the direction from the center of *overlap-region*$_{i,i+1}$ to the center of *free-region*$_{i-1,i}$. The overlapping region between the shifted *overlap-region*$_{i,i+1}$ and the original *overlap-region*$_{i,i+1}$ [8] is called the *shrink region*. The shift distance of *overlap-region*$_{i,i+1}$ is called the *shrink step*. The *overlap-region*$_{i,i+1}$ shifts by the *shrink step* each time until the shifted *overlap-region*$_{i,i+1}$ does not overlap the original *overlap-region*$_{i,i+1}$. In Figure 4.17 (a) (b) and (c), three *shrink regions* are generated. If *free-region*$_{i-1,i}$ is not totally included in the original *overlap-region*$_{i,i+1}$, the *minimum shrink region* is the *shrink region* with minimum area as shown in Figure 4.17 (c). If *free-region*$_{i-1,i}$ is totally included in the original *overlap-region*$_{i,i+1}$, the *minimum shrink region* is the minimum *shrink region* which overlaps *free-region*$_{i-1,i}$. In Figure 4.17 (d), the *overlap-region*$_{i,i+1}$ overlaps with *free-region*$_{i-1,i}$. In Figure 4.17 (e), the shifted *overlap-region*$_{i,i+1}$ does not overlap with *free-region*$_{i-1,i}$. The *shrink region* in Figure 4.17 (d) is the *minimum shrink region.*

The *free-region*$_{i,i+1}$ is found by shrinking *overlap-region*$_{i,i+1}$ toward *free-region*$_{i-1,i}$ if the corresponding swept volume of *overlap-region*$_{i,i+1}$ collides with objects in the environment, as illustrated in Figure 4.18 (a). In Figure 4.18 (b), the corresponding swept

[7] In the above example, we redefined m_k as p_{i+1}.

[8] The original *overlap-region*$_{i,i+1}$ is the *overlap-region*$_{i,i+1}$ without shift.

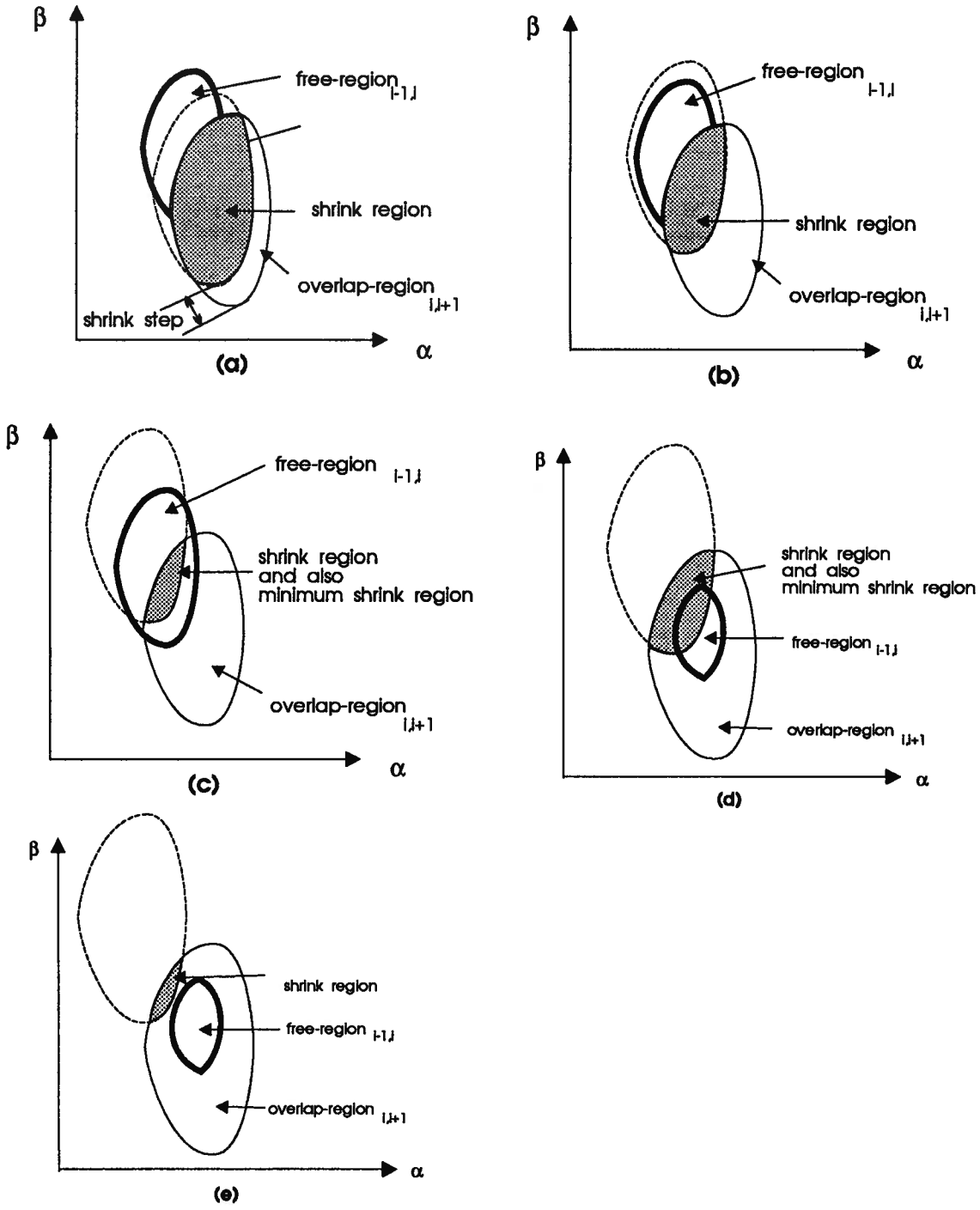

Figure 4.17: The *shrink step*, *shrink region* and *minimum shrink region*

volume of the *shrink region* again collides with objects in the environment. In Figure 4.18 (c), the corresponding swept volume of the *shrink region* is collision-free and this *shrink region* corresponds to *free-region*$_{i,i+1}$. If *free-region*$_{i,i+1}$ is not found when *minimum shrink region* has been checked, a midpoint is added between p_i and p_{i+1}. Adding a midpoint changes the overlapping region between *local* θ *region*s at successive points and allows some new configurations to be considered. A *minimum step size* is specified, and if *free-region*$_{i,i+1}$ is not found and the step size between successive points is smaller than the *minimum step size*, the current path is deleted from the *path list* and other paths in the *path list* are planned. If the *path list* is empty, the trajectory planning stops.

If *free-region*$_{i,i+1}$ is found, a search for the corresponding *free-range*$_{i,i+1}$ is carried out. The γ *range* is decomposed as a binary tree structure, and *free-range*$_{i,i+1}$ is searched using a breadth-first search. Only the γ ranges that has a common range with *free-range*$_{i-1,i}$ are searched. The *free-range*$_{i-1,i}$, *free-range*$_{i,i+1}$ and their common range contribute to the path of the tool handle. If no *free-range*$_{i,i+1}$ is found, the corresponding *free-region*$_{i,i+1}$ is invalid and must be shrunk by one *shrink step*, and the *free-range*$_{i,i+1}$ is searched again for the new *free-region*$_{i,i+1}$.

The Swept Volumes Used in Translation and Rotation Check

Three kinds of swept volume between curve points p_i and p_{i+1} are used in *Translation and Rotation Check*, the θ*-trans swept volume*, the γ*-trans swept volume* and the *shrink swept volume* which correspond to *overlap-region*$_{i,i+1}$, the γ *range* and *shrink region* respectively. The θ*-trans swept volume* is generated by extruding the intersection of two full range θ ***swept volume*s** in the direction of tool motion, i.e. the direction from p_i to p_{i+1}, as illustrated in Figure 4.19 (a). With reference to Figure 4.19 (a) (b), the size of the intersection volume depends on the angles ϕ_1 and ϕ_2 between the optimal orientations of the full range θ *swept volume*s at the points p_i and p_{i+1}. The length of the θ*-trans swept*

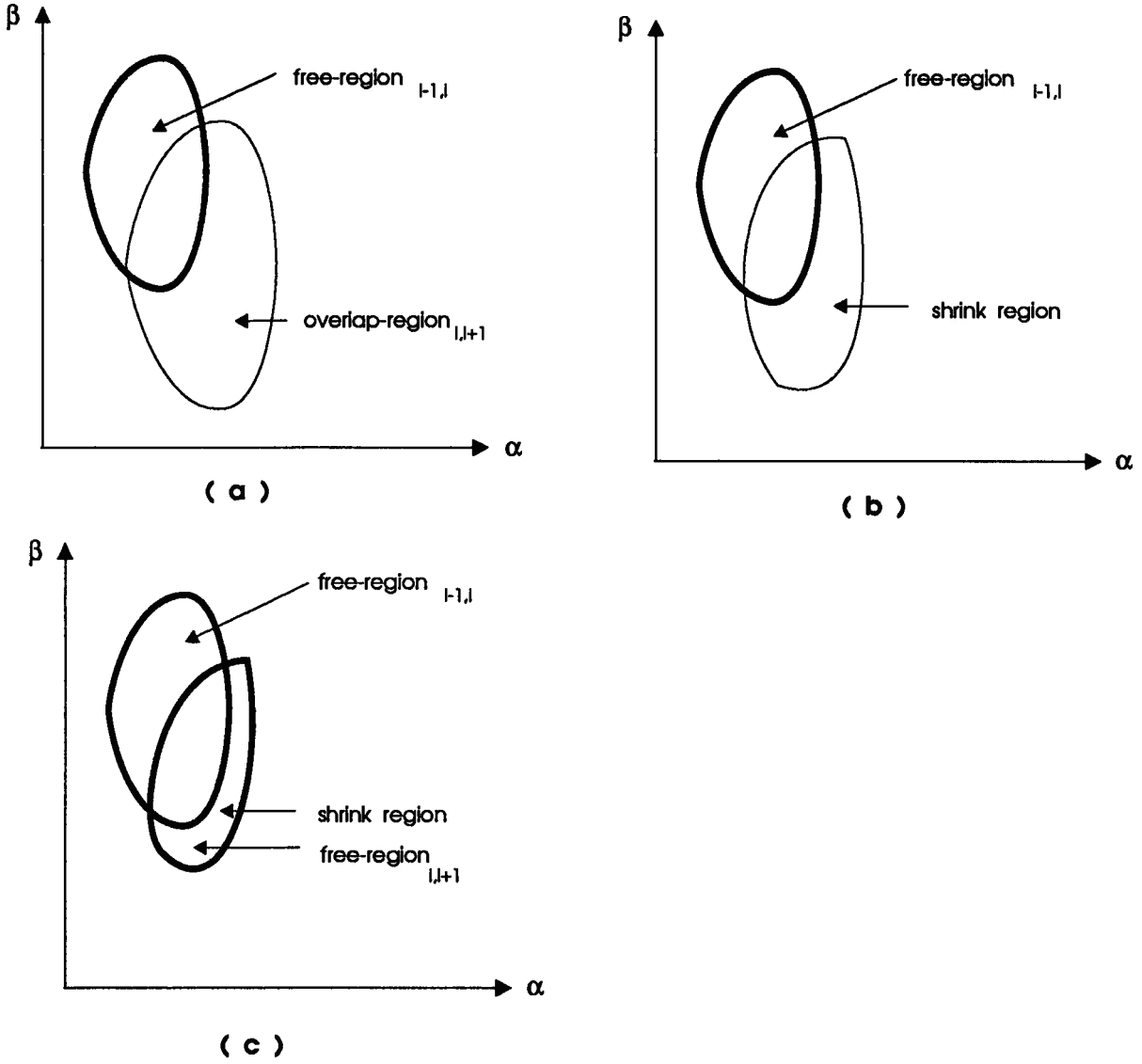

Figure 4.18: Shrinking *overlap-region*$_{i,i+1}$ toward *free-region*$_{i-1,i}$ in order to find *free-region*$_{i,i+1}$

volume corresponds to the distance between successive track points. With reference to Figure 4.19 (c) (d), the *intersection orientations* are defined as the orientations at the intersection points of the two θ *regions*. The minimum angle ω between the direction of tool motion and the plane defined by the two *intersection orientations* can assume any value, as illustrated in Figure 4.19 (d). The angle ω determines the shape of the θ*-trans swept volume*. Two θ*-trans swept volumes* with the same size but different ω angles are shown in Figure 4.19 (e) and (f). The θ*-trans swept volumes* corresponding to different sizes, lengths and angles are generated in advance. During the check, a θ*-trans swept volume* is selected based on the overlapping size, distance between successive track points and the angle ω with respect to the direction of tool motion.

The γ*-trans swept volume* corresponding to a θ*-trans swept volume* is approximated by using two vertical θ *swept volumes* shown in Figure 4.20 (b) to displace the θ *swept volumes* used to generate the θ*-trans swept volume*, as shown in Figure 4.20 (a). The length L, the angle ϕ and the allowable angle range θ_o of the θ*-trans swept volume* are known. In Figure 4.20 (a), line segment *oa* and line segment *ob* are on the boundary of the intersection of θ *swept volumes* and are also on the plane defined by the axes of θ *swept volumes*. The distance between o and a, and the distance between o and b are L. The distance between a and b is D, and the angle between *oa* and *ob* is ζ.

Using the cosine law in triangle *oab*

$$D^2 = 2L^2 - 2L^2 cos\zeta \tag{4.2}$$

Since

$$\phi = 4\theta_o - 2\zeta$$

we get

$$\zeta = 2\theta_o - \frac{\phi}{2} \tag{4.3}$$

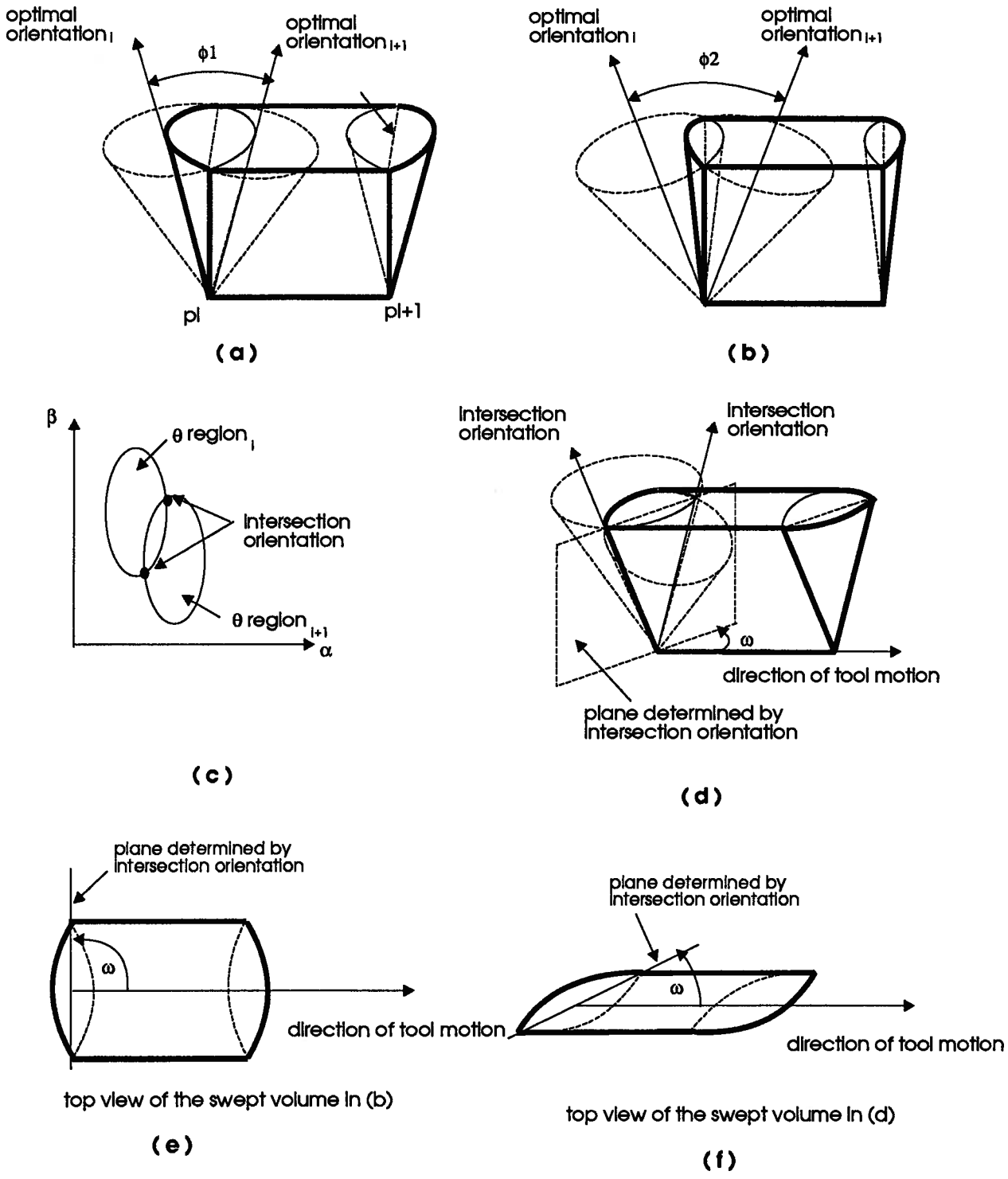

Figure 4.19: The *θ-trans swept volumes* used in *Translation and Rotation Check*

From Equation (4.2) and (4.3), we get

$$D = \sqrt{2}L\sqrt{1 - cos(2\theta_o - \frac{\phi}{2})} \tag{4.4}$$

In Figure 4.20 (b), d is the distance between the tips of the θ *swept volumes*.

$$d = 2Lsin\theta_o - D \tag{4.5}$$

From Equation (4.4) and (4.5), we get

$$d = 2Lsin\theta_o - \sqrt{2}L\sqrt{1 - cos(2\theta_o - \frac{\phi}{2})} \tag{4.6}$$

Given a θ*-trans swept volume*, the distance d between the two θ *swept volumes* can be calculated using Equation (4.6), and the intersection of two corresponding full range γ *swept volumes* is obtained as illustrated in Figure 4.20 (c). The corresponding full range γ*-trans swept volume* is generated by extruding the intersection along the direction of tool motion, as illustrated in Figure 4.20 (d). The length of the extrusion and the angle ω[9] are the same as those of the θ*-trans swept volume.* The full range γ*-trans swept volume* is decomposed into the binary tree structure illustrated in Figure 4.20 (e). Each γ*-trans swept volume* binary tree associates with the corresponding θ*-trans swept volume.*

Since the *shrink region* between p_i and p_{i+1} is the overlapping region between the shifted *overlap-region*$_{i,i+1}$ and the original *overlap-region*$_{i,i+1}$, the *shrink swept volume* is generated by taking the copy of the intersection of two θ *swept volumes*, called the *inter volume*, refer to Figure 4.21 (a), and rotating the *inter volume* ξ angle about the line which is in the plane defined by the two *intersection orientations* and is perpendicular to the line from the tip of the intersection to the center of the intersection, as shown in Figure 4.21 (b), and then extruding the intersection of the two *inter volumes* along the direction of tool motion, as illustrated in Figure 4.21 (c). Here ξ is the angle between the center of shifted *overlap-region*$_{i,i+1}$ and the center of original *overlap-region*$_{i,i+1}$.

[9] ω is the minimum angle between the direction of tool motion and the plane defined by the two *intersection orientations* as shown in Figure 4.19 (c).

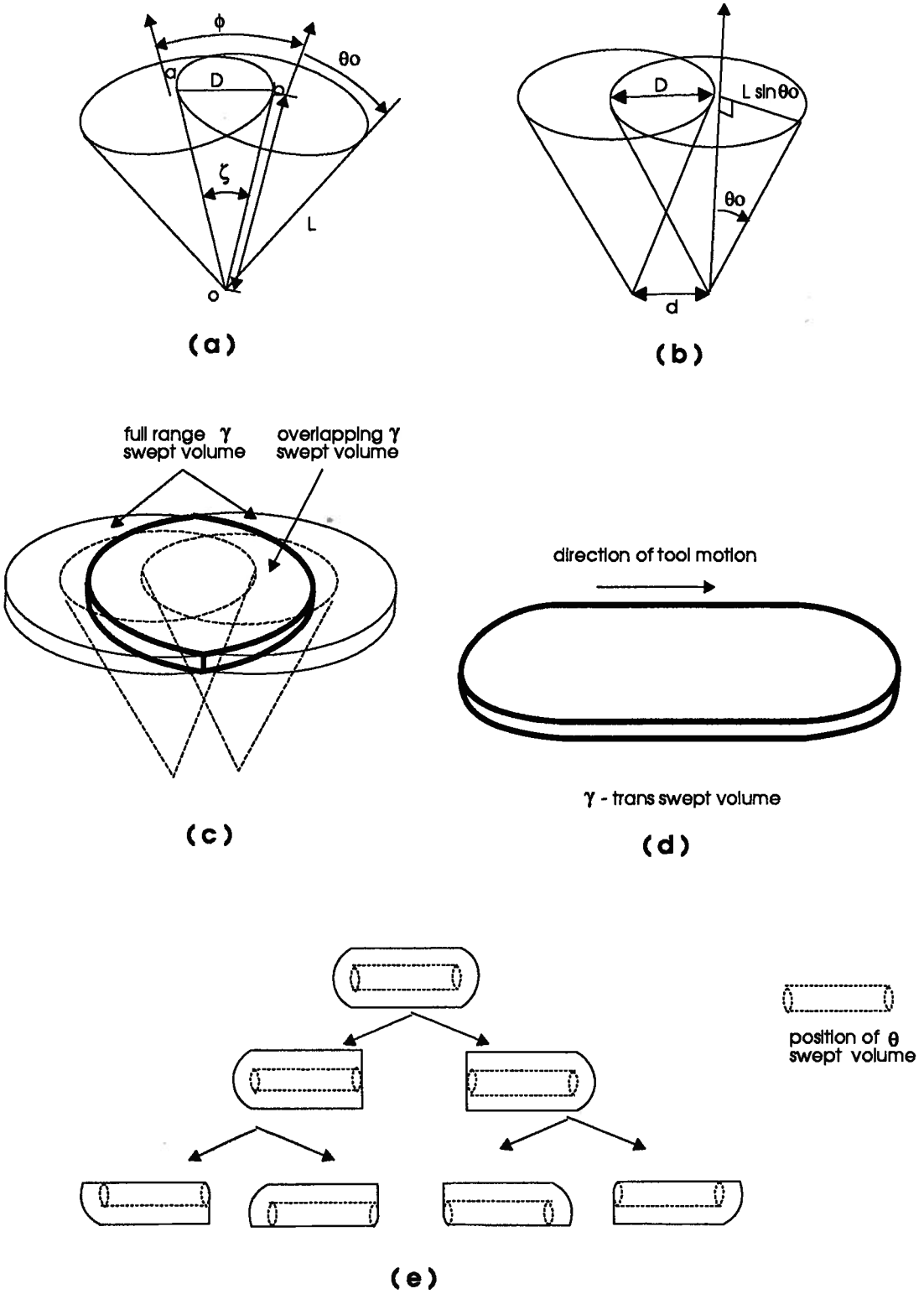

Figure 4.20: The *γ-trans swept volume* used in *Translation and Rotation Check*

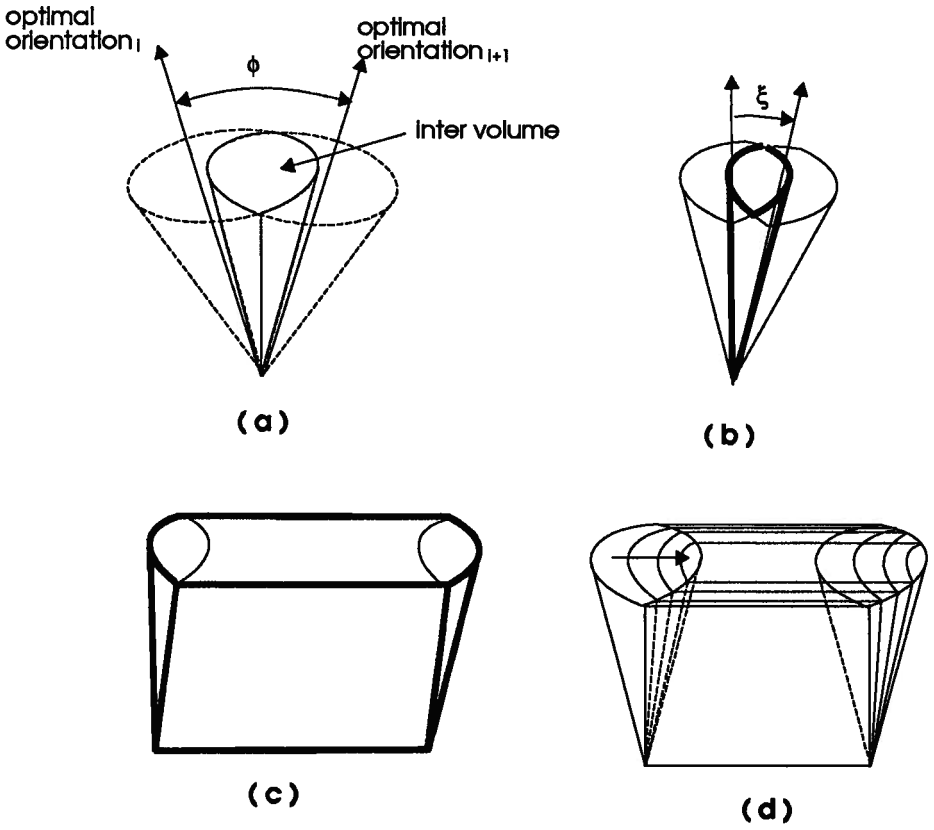

Figure 4.21: The *shrink swept volume* used in *Translation and Rotation Check*

The parameters in *Translate and Rotate Check* are *minimum step size* and *shrink step*. A large *minimum step size* results in a coarse search for a trajectory, while a small *minimum step size* increases the search resolution between successive track points. The *shrink step* determines shrinking resolution.

4.2.6 Inverse Kinematics Check

The motion of the robot tool is also limited by the robot joint constraints. Computation of the robot inverse kinematics solution requires the configuration and position of the robot wrist to be calculated from the position of the track point and the selected configuration of the robot tool. With reference to Figure 4.22, the following homogeneous coordinate frames are used in the computation:

1. Robot Base Frame F_r: the frame with origin at the robot base.
2. Track Point Frame F_p: the frame located at the track point having the same orientation as F_r.
3. Selected Configuration Frame F_s: the frame with an origin coincident with F_p but rotated by α_s about Y_p and β_s about Z_p.
4. Tool Frame F_t: the frame with an origin coincident with F_s but rotated by γ about Z_s.
5. End Effector Frame F_e $(\vec{a}, \vec{n}, \vec{s})$: $\vec{a}$ and $\vec{n}$ are in the plane determined by the tool axis and the tool handle, and $\vec{a}$ points to the track point.

The homogeneous transformations [Wolo 91] $H_{r,p}$, $H_{p,s}$, $H_{s,t}$ and $H_{t,e}$ defining the robot position and orientation can be obtained from the frame definitions listed above. The robot base to tool configuration coordinate transformation $H_{r,e}$ is calculated as,

$$H_{r,e} = H_{r,p} H_{p,s} H_{s,t} H_{t,e} \tag{4.7}$$

The right hand side of Equation (4.7) corresponds to the homogeneous transformations in the sequence of the robot base, tool tip, tool head, tool handle, and end effector. Given a tool configuration (α_s, β_s, γ, x, y, z), the right hand side of Equation (4.7) is known. The left hand side of Equation (4.7) represents the homogeneous transformations in the sequence of the robot base, link1, link2,..., end effector. For a given robot, the final overall (base to tool) configuration coordinate transformation $H_{r,e}$, which contains the required joint angles, is fixed [Wolo 91]. So the robot joint angles can be calculated in Equation (4.7). The details are discussed in Appendix C.

The configurations in the path comprising *free-region*$_{i-1,i}$ and *free-region*$_{i,i+1}$ are determined as follows:

1. The configurations in *free-region*$_{i-1,i}$ are the configurations of the robot tool at the midpoint between p_{i-1} and p_i.

2. The configurations in the overlapping region between *free-region*$_{i-1,i}$ and *free-region*$_{i,i+1}$ are the configurations of the robot tool at p_i.

3. The configurations in *free-region*$_{i,i+1}$ are the configurations of the robot tool at the midpoint between p_i and p_{i+1}.

Multiple inverse kinematics solutions can be computed based on one position/orientation of the robot tool. For instance, eight solutions are obtained for the PUMA 560 robot. If all joint angles in an inverse kinematics solution fall within the robot joint constraints, the inverse kinematics solution is valid.

In each region, only the configurations which are near the center of a region are used to compute the inverse kinematics solutions. This provides a degree of safety in the trajectory since there is a collision-free region between the selected configuration and the environment. If none of the configurations produce a valid inverse kinematics solution,

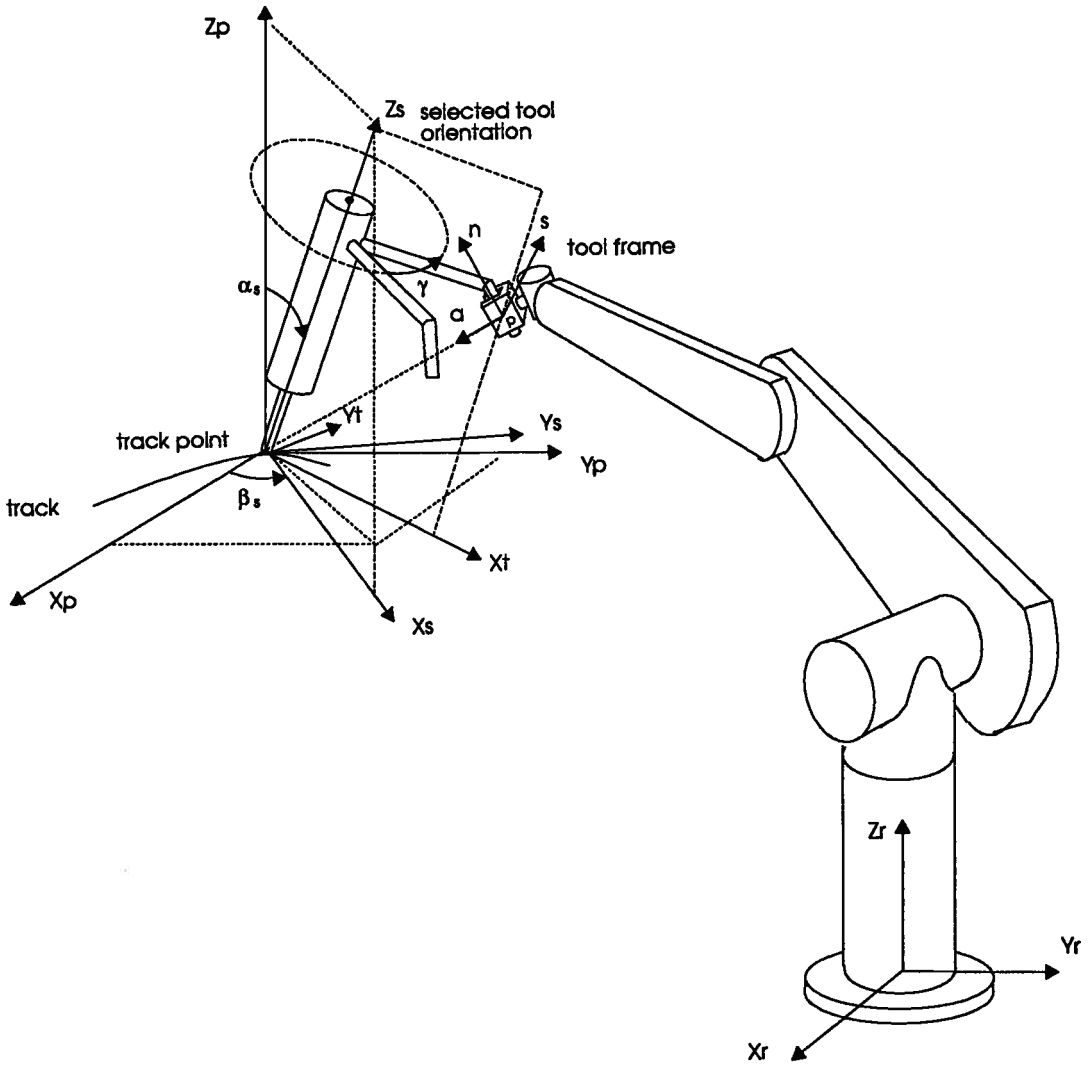

Figure 4.22: A series of coordinate frames

this check fails and other paths in the *path list* are explored. If valid inverse kinematics solutions are computed, these solutions and the corresponding configurations are stored at a subsequent stage in the planning. A cost function is used to select the inverse kinematics solutions and their corresponding configurations at each point to generate the optimal trajectory. Details of this search are discussed in Section 4.2.8.

The number of configurations selected for the inverse kinematics computation is very important. If just a few configurations are selected, the check may fail although there may be unexplored configurations that correspond to valid inverse kinematics solutions. Increasing the number of configurations results in a better chance of finding valid inverse kinematics solutions, and a better trajectory will be planned since more configurations and inverse kinematics solutions can be used by the cost function. This, however, takes more computation time and requires more memory to store the data.

4.2.7 Control the Path Planning Using A^* Search Algorithm

The path consists of a list of *free-regions* at the curve points which have been included in the path. If there is more than one path in the *path list*, an A^* search algorithm [Wins 84] is used to control the path planning for multiple paths. This search is summarized as follows,

Until the *path list* is empty, determine if the first path in the *path list* includes the final point on the curve,

1. If the first path includes the final curve point, do nothing.

2. If the first path does not include the final curve point,

   (a) Remove the first path from the *path list.*

   (b) Form new paths from the removed path by extending one step.

(c) Add the new paths to the *path list.*

(d) Sort the *path list* in the order of increasing cost, where the cost is the accumulated *PathCost* so far plus a lower-bound estimate of the cost remaining.

(e) If two or more paths reach a common curve point, delete all those paths except for the path that reaches the common curve point with the minimum cost.

The *PathCost* is calculated as

$$PathCost = W_V C_V + W_O C_O - W_P C_P \tag{4.8}$$

Where W_V, W_O and W_P are the weights of cost C_V, C_O and C_P respectively.

C_V, the cost measuring the average area of the path, is calculated as

$$C_V = \sum_{i=1}^{n} \frac{V_i}{n} \tag{4.9}$$

where V_i is the volume of *free-cell*$_{i,i+1}$ and n is the number of current track points. This cost is used to select a wider path.

C_O, the cost measuring the average departure of the path segment away from the optimal orientation, is calculated as

$$C_O = \sum_{i=1}^{n} \frac{\sqrt{(\alpha_{di} - \alpha_i)^2 + (\beta_{di} - \beta_i)^2}}{n} \tag{4.10}$$

where α_i is the average of α_{min} and α_{max} in the *free-region*$_{i,i+1}$, β_i is the average of β_{min} and β_{max} in the *free-region*$_{i,i+1}$, and α_{di} and β_{di} are the averages of α and β angles in the optimal orientations at p_i and p_{i+1} respectively. This cost is used to select the path which is closer to the optimal orientation.

C_P, the cost measuring the percent of the curve that has been travelled, is calculated as

$$C_P = \frac{n_c}{n} \tag{4.11}$$

where n_c is the current point number and n is the number of track points travelled. This cost is used to select the path which is close to the end of the curve.

The parameters in this check are the weights in the cost function. By changing the weights, the trajectory is planned in different ways. For example, a large value of W_V means a wider path is preferred, a large value of W_O means that the application constraint is important, and a large value of W_P means the speed of finding a path is important. The selection of the weights is discussed in 5.4.

4.2.8 Optimize Trajectory

In each path segment, the inverse kinematics solutions of a number of tool configurations are calculated as described in Section 4.2.6. After finding the path along the curve, the optimal trajectory along the curve is obtained using a second A^* search to select both the robot inverse kinematics solutions and corresponding tool configurations from the path. The A^* search is discussed in Section 4.2.7. The cost used in the second search is calculated as

$$\begin{aligned} TrajCost_k &= W_J \sum_{i=1.0}^{k} C_{Ji} + W_D \sum_{i=1.0}^{k} C_{Di} + W_H \sum_{i=1.0}^{k} C_{Hi} \\ &\quad + \sum_{j=1}^{J} (W_{Rj} \sum_{i=1.0}^{k} C_{Rij}) + W_C \sum_{i=1.0}^{k} C_{Ci} - W_P C_P \end{aligned} \tag{4.12}$$

Where $1.0 \leq k \leq n$, k is current track point number, n is the final track point number in the track, and J is the number of robot joints. Since some midpoints are added in the track, the track point index (k) is not an integer.

In Equation (4.12), W and C represent weights and costs respectively. C_{Ji} is the cost measuring the joint angle displacement from the midpoint of the valid joint angle. If several joint angles are available, the joint angle closest to the midpoint of joint range is preferred. This avoids joint boundaries as the tool travels the curve. The cost C_{Ji} is

calculated as

$$C_{Ji} = \sum_{j=1}^{J} \frac{(\frac{\theta_j - \theta mj}{\theta_{fj}})^2}{6} \tag{4.13}$$

where θ_j and θmj are the jth joint angle and the midpoint of jth joint range, and θ_{fj} is the full range of the jth joint.

C_{Di}, the cost measuring deviation of the tool orientation from the optimal orientation, is calculated as

$$C_{Di} = \frac{(\frac{\alpha_{oi} - \alpha_i}{\alpha_{Li}})^2 + (\frac{\beta_{oi} - \beta_i}{\beta_{Li}})^2}{2} \tag{4.14}$$

where α_{oi} and β_{oi} are the α and β values of the optimal orientation at the ith track point, α_i and β_i are the α and β values of the selected orientation at the ith track point, and α_{Li} and β_{Li} are the α and β ranges of the *Local C-space* at the ith track point.

C_{Hi}, the cost measuring the difference between the selected configurations at the ith and $i-1$th track points is calculated as

$$C_{Hi} = \frac{(\frac{\alpha_i - \alpha_{ip}}{\alpha_{Li}})^2 + (\frac{\beta_i - \beta_{ip}}{\beta_{Li}})^2 + (\frac{\gamma_i - \gamma_{ip}}{\gamma_{Li}})^2}{3} \tag{4.15}$$

where α_i, β_i and γ_i are the α, β and γ values of selected tool configuration at the ith track point, α_{ip}, β_{ip} and γ_{ip} are the α, β and γ values of the selected tool configuration at the $i-1$th track point, and α_{Li}, β_{Li} and γ_{Li} are the α, β and γ ranges of the *Local C-space* at the ith track point.

C_{Rij}, the cost measuring the difference between the robot joint angles at the ith and $i-1$th track points, is calculated as

$$C_{Rij} = \frac{1}{n_j} \sum_{i=1}^{n_j} (\frac{\theta_{j,i} - \theta_{j,ip}}{\theta_{j,f}})^2 \tag{4.16}$$

where n_j is the joint number, $\theta_{j,i}$ and $\theta_{j,ip}$ are the jth joint angles at the ith and $i-1$th track point respectively, and $\theta_{j,f}$ is the full range of the jth joint angle.

C_{Ci}, the cost measuring the change of robot configuration at the ith and $i-1$th track points, is calculated as

$$C_{Ci} = (I_i - I_{ip})^2 \tag{4.17}$$

where I_i and I_{ip} are the robot configuration indices at the ith track point and $i+1$th track point respectively.

C_P, the cost measuring the percent of the curve that has been travelled, is defined by Equation (4.11).

The cost function weights are used to bias the trajectory as shown in following table,

Weights	Effect on trajectory planning
W_J	safe robot motion
W_D	maintaining optimal tool orientation
W_{Hi} and W_{Rj}	smooth motion
W_C	robot configuration avoidance
W_P	speed of search optimal trajectory

Normally, a large value of W_C is applied to keep the same robot configuration along the curve. If the robot configuration changes from "elbow up" to "elbow down", or changes from "right" to "left", as illustrated in Figure 4.23, discontinuous motion of robot disrupts the smooth motion of the tool. If a robot configuration selected at the beginning of search changes in the middle of the curve, all proposed trajectories using this configuration will have a very high cost. Details of the selection of weights for an example application of this algorithm are given in Chapter 5.

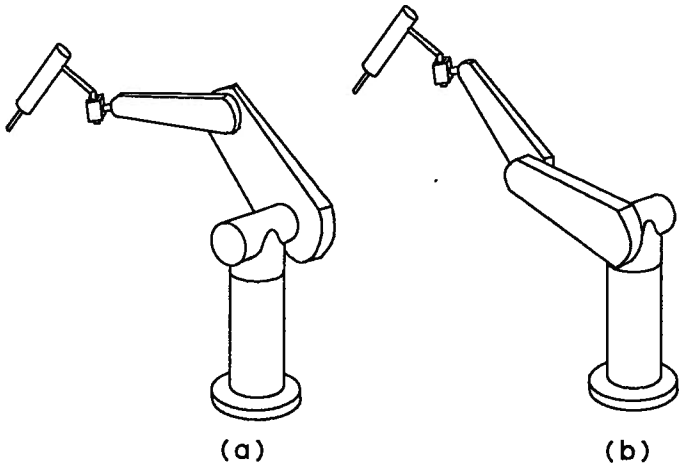

Figure 4.23: Different robot configurations

4.3 Summary of the Midpoint Algorithm

The Midpoint Algorithm includes the following steps:

1. Reconstruct the curve with Catmull-Rom splines.

2. Predetermine the optimal orientation at each curve point according to the geometric criterion of the robot tool specified by the application.

3. Predetermine the parameters in trajectory planning according to the requirements of applications.

4. Store the curve point information into a linked list.

5. Initialize the *path list* by using the *Point Check* procedure at the first point. If the *path list* is empty, the path planner stops.

6. Use the A^* search algorithm to explore multiple paths in the *path list.* The minimum cost path is extended to the next point. The extension of a path between successive

points is subject to the following checks: *Curvature Check*, *Overlap Check*, *Translation and Rotation Check*, *Inverse Kinematics Check* and *Path Cost Check*. The track of tool tip is built and points may added or deleted along the track.

7. If no path though the track can be found after trying all the path in the *path list*, the path planner stops.

8. If the minimum cost path in the *path list* is found, an A^* search selects the tool configurations and robot joint values in the final path.

The flowchart of the Midpoint Algorithm is illustrated in Figure 4.24.

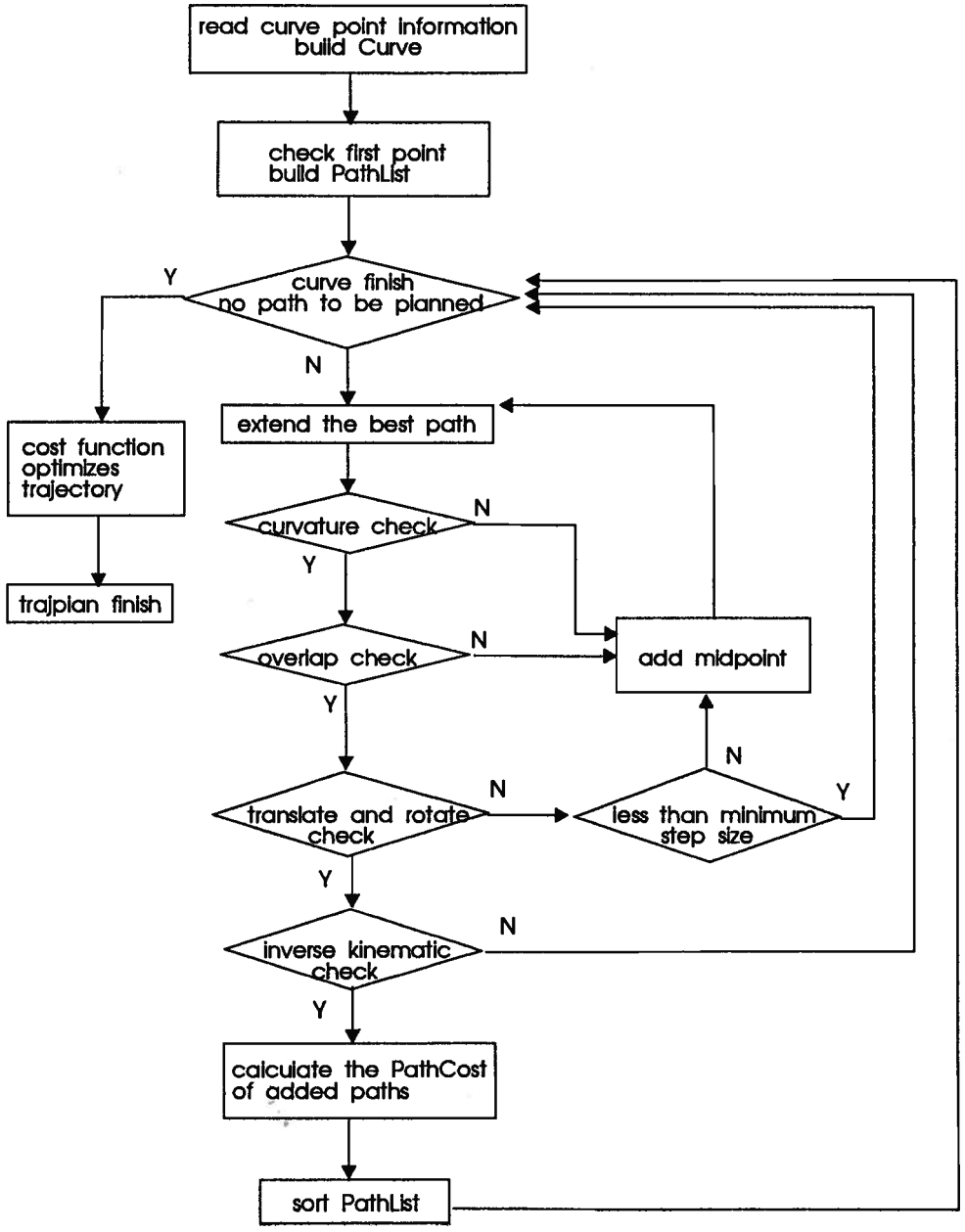

Figure 4.24: Flowchart for the Midpoint Algorithm

Chapter 5

Computer Implementation and Examples

5.1 Overview

An off-line simulation program, TRAJPLAN, has been developed in order to test the Midpoint Algorithm. Using TRAJPLAN, the user only needs to input the application specification, to select the curves to be processed and to indicate the environment objects that may collide with a robot tool during the process. A trajectory for the robot tool will then be automatically planned. The output of TRAJPLAN is a set of sequential joint angles which can be downloaded to the robot.

TRAJPLAN has been developed using Zortech C++ release 3.0, AutoCAD release 12, AutoCAD Advanced Modeling Extension (AutoCAD AME) release 2.1, and AutoCAD Development System (AutoCAD ADS). As a solid (3-D) and region (2-D) modeling program, AutoCAD AME is used to generate the swept volumes of the robot tool. AutoCAD ADS is a C language programming environment for developing AutoCAD applications. In TRAJPLAN, the AutoCAD ADS is used to move the swept volumes of tool and to check for interference between swept volumes and the environment.

Since TRAJPLAN only deals with two adjacent points on a curve, the complexity of the curve and environment does not have any effect in the algorithm. A practical limitation to TRAJPLAN is that the amount of data to be stored may exceed the system memory model if many curve points are used. This has not been a problem in the trials of the system to date where there are typically 100 points have been selected. TRAJPLAN

is compiled using the Zortech C++ 3.0 compiler using the P memory model (Phar Lap 386—DOS Extender, a full 32 bit 386 DOS Extender) to allow the program to operate in 32 bit protected mode with a linear address space of 4 Gbytes.

While PUMA 560 and CRS A460 inverse kinematics modules have been developed to work with TRAJPLAN, kinematic modules for other robots can be easily linked with TRAJPLAN. TRAJPLAN has successfully demonstrated automatical programming of a robot welding system and a fish butchering system.

5.2 The Simulation Program TRAJPLAN

The simulation program TRAJPLAN is initiated by identifying the curves to be processed and the environment objects which may collide with the robot tool. The user still needs to select the desired way to plan the trajectory (refer to Section 5.4), precision level of trajectory (refer to Section 5.5), and the optimal orientation (refer to Section 5.6). If no selection is given, default settings are used.

After the user initiates the TRAJPLAN, the path planning starts by reconstructing the identified curve based on sample curve points using the Catmull-Rom Spline method. A linked list is used to store the reconstructed curve. The advantage of the linked list is that a node can be easily inserted and deleted. The path planning procedure builds and updates the *path list*, a linked list which records all of the potential paths. Each path in the *path list* is also a linked list which links the path segments between successive points. Since different paths may pass through different environments and add different midpoints into the curve, each path must record its points. The *path list* is updated with following steps:

STEP 1 Use check loop, including *Curvature Check*, *Overlap Check*, *Translation and Rotation Check*, *Inverse Kinematics Check* and *Path Cost Check*, to extend the top

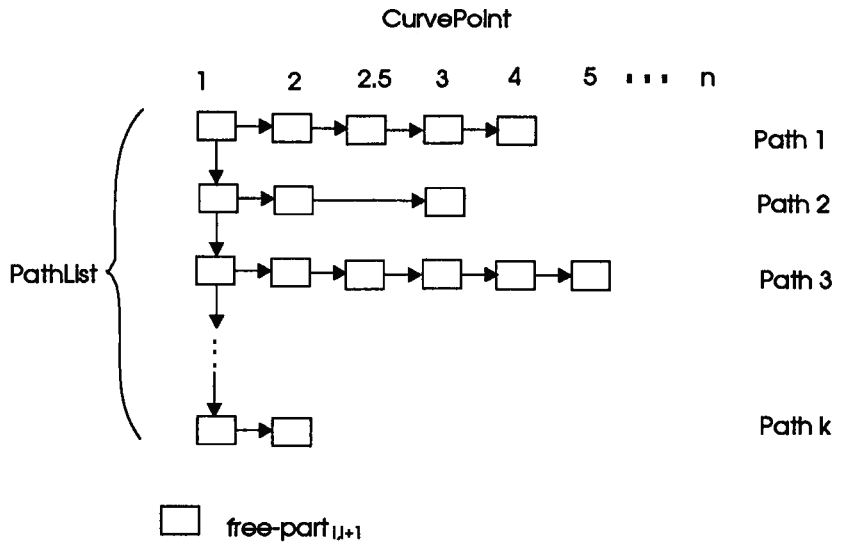

Figure 5.1: PathList data structure

path in the *path list.* If the check loop fails, add a midpoint and repeat STEP 1.

STEP 2 Substitute top path by extended path and sort the *path list* by the PathCost, placing the path with minimum cost at the top of the *path list.* Finish the path planning if the last curve point is reached. Otherwise, return to STEP 1.

One of the basic tasks of planning is to place swept volumes of the tool at a track point or between successive track points and to check for the interference between the swept volumes and environment in order to find a *free cell* or *free-cell*$_{i,i+1}$. At the first point, *free cells* are found by shrinking the θ *swept volumes* in order to obtain collision free θ *swept volumes*, and then using breadth-first search to find the collision free γ *swept volumes* in corresponding γ *swept volume* binary tree. If no collision free γ *swept volume* is found, the θ *swept volume* is shrunk again and the corresponding γ *swept volume* binary tree is searched. Between subsequent curve points, the search for *free-cell*$_{i,i+1}$ is very similar to the search for *free cell*, i.e. search θ*-trans swept volumes* first and then search γ*-trans swept volumes* corresponding to the collision free θ*-trans swept volume.* The procedure

is illustrated in Figure 5.2. In order to access the swept volumes efficiently during path planning, the swept volumes are down loaded into memory. This detail is discussed in Section 5.3.

For the PUMA 560 and CRS A460 robots, each robot tool configuration can be used to calculate eight robot inverse kinematics solutions. The trajectory of the robot tool has two parts: the configurations and positions of the tool, and the corresponding inverse kinematics solutions. If TRAJPLAN cannot find a path, or the final trajectory based on the selected path contains a change of robot configuration, TRAJPLAN reports the error or warning message to the user.

5.3 Down Load the Swept Volume Into Memory

Loading the swept volume into memory provides efficient access to the swept volumes during the interference check, which speeds up the trajectory planning. Loading all of the swept volumes into memory, however, results in a simple tree structure to be used, which cannot represent a configuration space with high resolution, even for *Local C-space.* It is impossible to load a huge data structure into memory due to the limitations on memory space and the restrictions of MS DOS. The θ *swept volumes* are stored in a one dimensional array and θ*-trans swept volumes* are stored in a three dimensional array based on the size, length and angle of the swept volume. Each θ swept volume has a full range γ swept volume associated with it. The γ swept volume is decomposed into a binary tree. If each dimension in the θ *swept volume* array or the θ*-trans swept volume* array contains three elements, and a five level γ swept volume binary tree [1] is associated with each array element, the total number of swept volumes is 930 $(3 \times 31 + 3 \times 3 \times 3 \times 31)$. MS DOS, however, only allows 250 files to be opened simultaneously. To solve this problem,

[1] A five level binary tree contains 31 nodes.

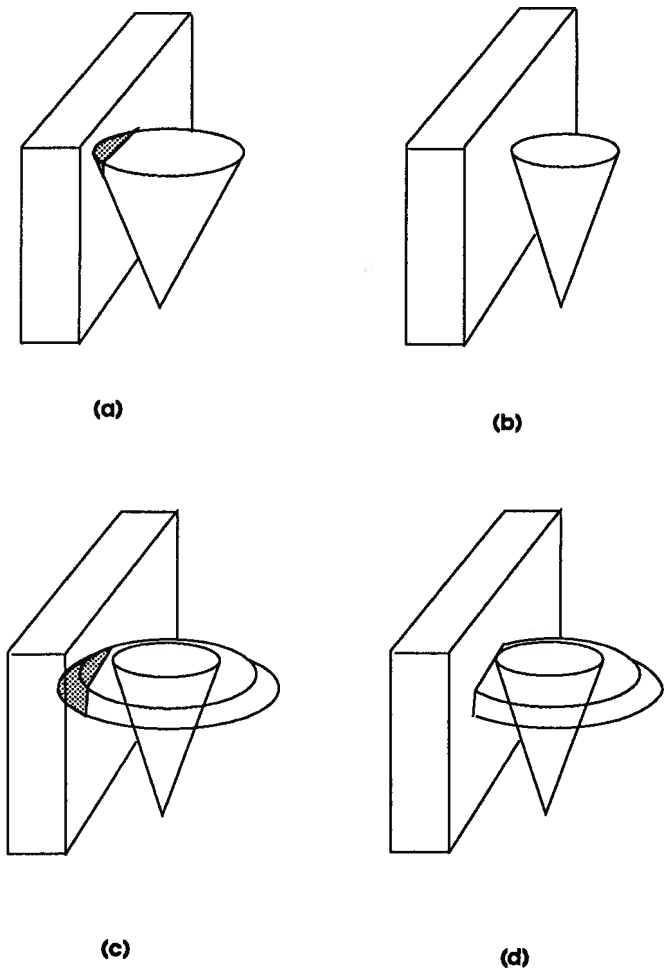

Figure 5.2: The procedure for searching for collision free swept volume (a) The θ *swept volume* collides with environment, (b) Shrinking θ *swept volume* for the collision free θ *swept volume*, (c) The γ *swept volume* collides with environment, (d) Search γ *swept volumes* for the collision free γ *swept volume.*

only the swept volumes with different shapes are loaded into memory. The remaining swept volumes can be obtained by rotating the loaded swept volume having the same shape. The rotation angle α, β and γ are determined by the difference in the α, β and γ angles between two corresponding swept volumes. In this way, 120 $(3 \times 4 + 3 \times 3 \times 3 \times 4)$ swept volumes instead of 930 swept volumes are loaded into memory, and the exploration of a high resolution configuration space is possible.

5.4 Plan the Trajectory as Desired

Several options for planning the trajectory can be selected by the user:

1. Select a trajectory which meets the application requirements as quickly as possible.
2. Select the optimal trajectory from all of the potential trajectories.
3. Select a trajectory which maintains the orientation close to the user specified orientation.
4. Select a trajectory which avoids a change of robot configuration.
5. Select a trajectory which provides for smooth motion of the tool.

If time is not of great concern, the user may require TRAJPLAN to provide the optimal trajectory. Or, if the user has limited time, he can require TRAJPLAN find a path as quickly as possible.

The weights in cost functions (Equation (4.8) and Equation (4.12)) determine the options of TRAJPLAN. The default values of the weights to calculate *PathCost* are

shown in following table,

Weight	Effect on trajectory planning	Value	Comments
W_P	speed of planning path	0.6	the most important
W_O	path close to optimal orientations	0.3	important
W_V	width of path	0.1	not very important

The default values of the weights to calculate *TrajCost* are shown in following table[2],

weight	effect aspect in trajectory	value	comments
W_J	joint angle is close to the center of joint angle range	0.01	not very important
W_D	trajectory is close to optimal orientation	0.15	important
W_H	trajectory keeps its history	0.2	important
W_P	speed of planning path	0.15	important
W_{R1}	the first joint angle keeps its history	0.15	important
W_{R2}	the second joint angle keeps its history	0.15	important
W_{R3}	the third joint angle keeps its history	0.1	important
W_{R4}	the fourth joint angle keeps its history	0.03	not very important
W_{R5}	the fifth joint angle keeps its history	0.03	not very important
W_{R6}	the sixth joint angle keeps its history	0.03	not very important
W_C	the change of robot configuration	100	very important

A large value is assigned to W_C in order to avoid the robot arm inversion. The values of W_{R1}, W_{R2}, W_{R3}, W_{R4}, W_{R5} and W_{R6} indicate that the motion of the robot wrist is

[2]A large value is assigned to W_C in order to avoid robot arm inversion.

more flexible than the motions of upper arms.

5.5 The Precision of Trajectory

The precision of trajectory is very important since it affects the quality of process. Different applications have different precision requirements. For example, a welding application may require the tool tip to follow the curve more strictly than a painting application. The higher the required precision, the more careful the motion, although the precision should be determined by the user according to the requirement of application. Otherwise the following default values are assumed,

d_{min}	0.4 inch
d_{max}	0.8 inch
number of intermediate points	8
minimum step size	0.02 inch
number of configurations picked up from path segment	24

For the convenience of inexperienced users, the values of parameters have been set at different accuracy levels. If the default accuracy level is not suitable for the application, the user can simply select the accuracy level to alter the speed or precision of the trajectory planning algorithm.

5.6 The Specification of Optimal Tool Orientation

To use TRAJPLAN, user need to input the application specification including the optimal orientation of robot tool relative to the curve, the allowable deviation of the tool orientation away from the optimal orientation, and the allowable deviation in the distance between the tool tip and the curve. Without the input, TRAJPLAN will plan

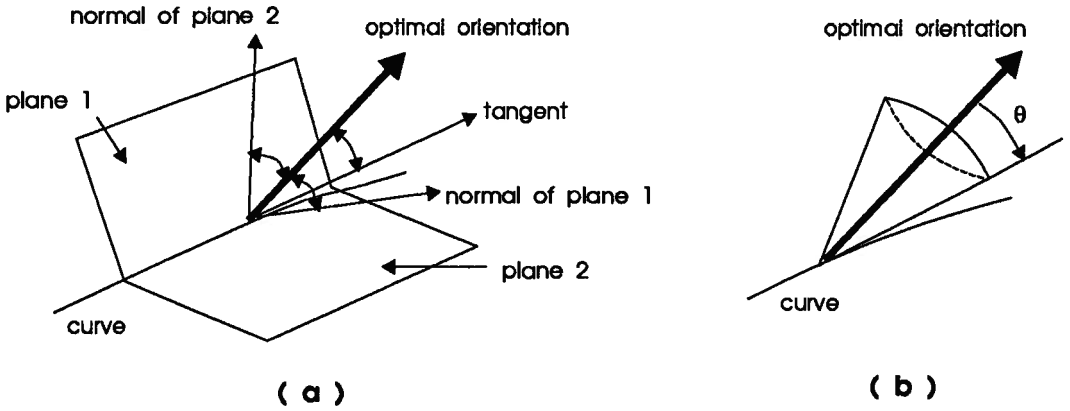

Figure 5.3: The optimal orientation of robot tool

a trajectory based on the default specifications. With reference to Figure 5.3 (a), the default geometric criterion of the tool optimal orientation at a curve point is defined as follows,

1. The tool optimal orientation is perpendicular to the tangent of curve at the point.
2. The tool optimal orientation corresponds to the vector that bisects the normals of the two planes on each side of the curve.
3. The tolerance of the orientation deviation θ from the tool optimal orientation is 30° as shown in Figure 5.3 (b).

5.7 Examples: Automatic Welding and Fish Butchering Processing

5.7.1 Automatic Robot Welding Process

An automatic robot welding system is shown in Figure 5.4. The process constraints include the distance between the welding head and the seam, the optimal welding orientation, collision avoidance, valid robot inverse kinematic solutions, and robot arm inversion avoidance. Complete knowledge of the system including the position of objects

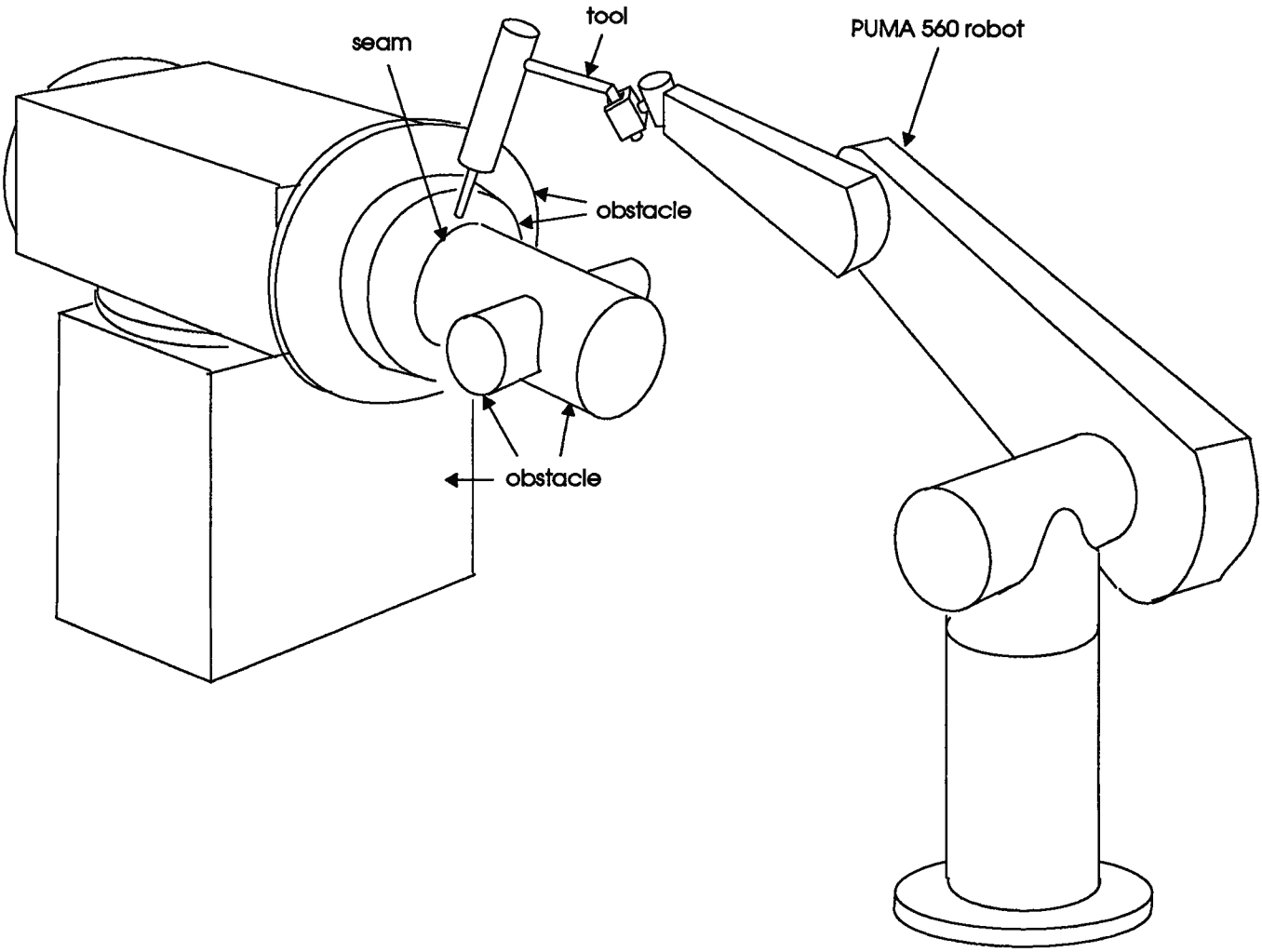

Figure 5.4: The automatic robot welding workcell

in the workcell and the seams is provided by a computer-aided design system, AutoCAD. In the workcell shown in Figure 5.4, TRAJPLAN finds a collision free path subject to process constraints in 1'20". Most of this time is taken by the AutoCAD interference checks. An example procedure for searching for collision free paths by placing various swept volumes on the seam is illustrated in Figure 5.5[3] and described as follows,

(a) Place the full range θ *swept volume* at the first seam point. The θ *swept volume* collides with obstacles.

(b) Shrink the θ *swept volume* until it does not collide with obstacles.

(c) Place the full range γ *swept volume* corresponding to the selected θ *swept volume* at the first seam point. The γ *swept volume* collides with obstacles.

(d) Search the γ *swept volumes* at lower levels in the γ *swept volume* binary tree. The first γ *swept volume* selected collides with obstacles.

(e) Select another γ *swept volume.* This γ *swept volume* is collision free.

(f) Test the θ*-trans swept volume* corresponding to the selected collision free θ *swept volume.* The swept volume is collision free.

(g) Place the full range γ*-trans swept volume* corresponding to the selected θ*-trans swept volume* on the seam. The γ*-trans swept volume* collides with obstacles.

(h) Search the γ*-trans swept volumes* at lower levels in the γ*-trans swept volume* binary tree. The selected γ*-trans swept volume* collides with obstacles.

(i) Select another γ*-trans swept volume.* This γ*-trans swept volume* also collides with obstacles.

[3]The dash line is the seam and the cylinder and the plate are the obstacles.

(j) Search the *γ-trans swept volumes* one level down in the *γ-trans swept volume* binary tree. The selected *γ-trans swept volume* collides with obstacles.

(k) Select another *γ-trans swept volume.* The selected *γ-trans swept volume* is collision free.

(l) Select another *γ-trans swept volume.* The selected *γ-trans swept volume* collides with obstacles.

(m) Select another *γ-trans swept volume.* The selected *γ-trans swept volume* is collision free. Up to now, there are two possible paths found in (k) and (m) respectively.

5.7.2 Automatic Fish Butchering Process

The Industrial Automation Laboratory at UBC has developed a system for automatically selecting cutting contours for a fish butchering process [Gama 93]. The fish butchering workcell is shown in Figure 5.7. When fish passes along the convey or the position and the shape of fish contour is calculated by a knowledge-based computer image processing system. The process constraints include the distance between fish and a water-jet cutter, the optimal cutter orientation, collision avoidance, valid robot inverse kinematic solutions, and robot arm inversion avoidance. Using the workcell shown in Figure 5.7, TRAJPLAN finds a collision free path subject to process constraints in 15 seconds. An example procedure for searching for collision free paths by placing various swept volumes on the fish contours is illustrated in Figure 5.6[4] and described as follows,

(a) Place the full range *θ swept volume* at the first contour point. The *θ swept volume* does not collide with obstacles.

[4]The dash line is the fish contour.

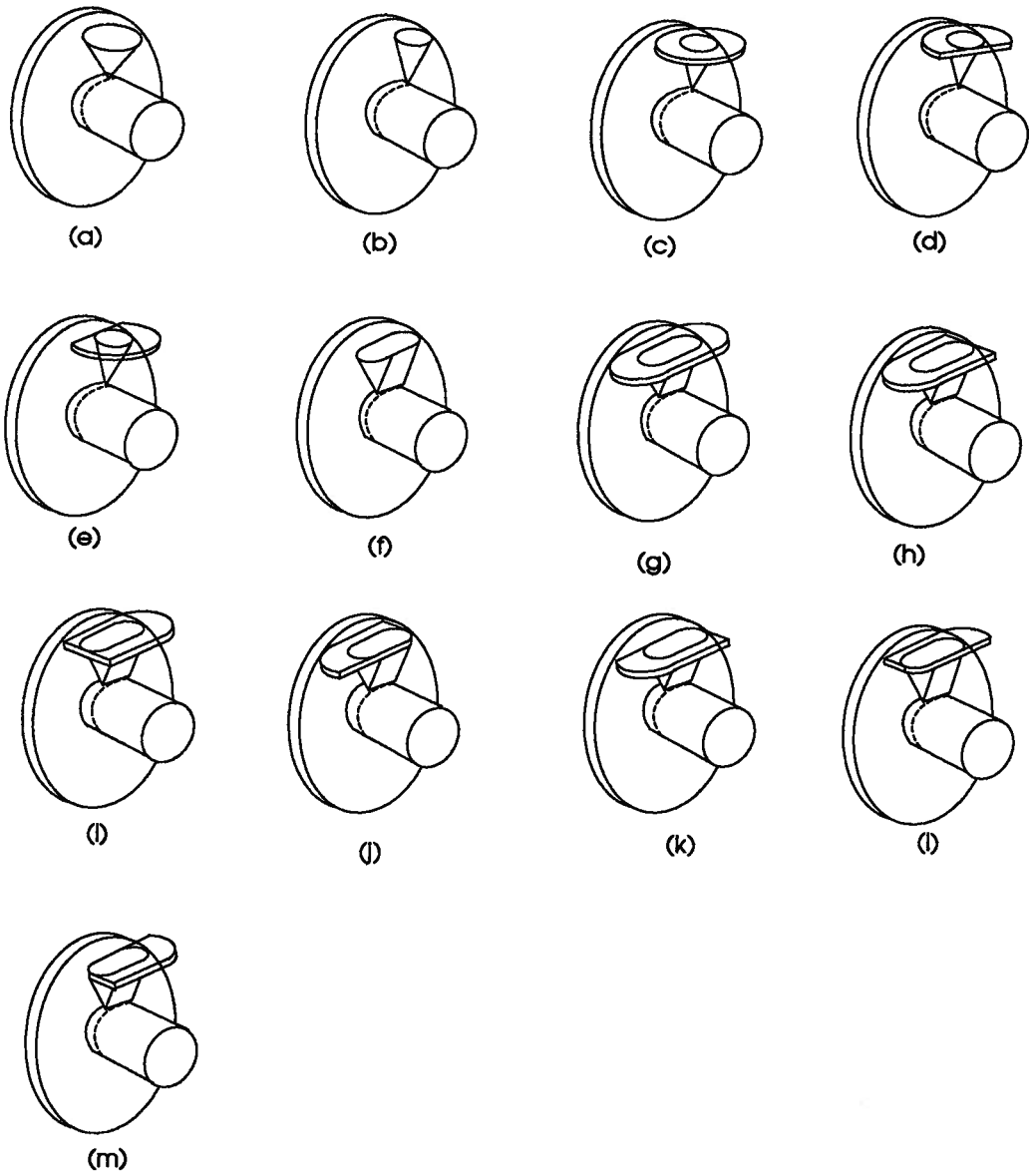

Figure 5.5: Procedure for searching for collision free paths.

(b) Place the full range γ *swept volume* corresponding to the selected θ *swept volume* at the first contour point. The γ *swept volume* collides with obstacles.

(c) Search the γ *swept volumes* at lower levels in the γ *swept volume* binary tree. The first γ *swept volume* selected collides with obstacles.

(d) Select another γ *swept volume.* This γ *swept volume* is collision free.

(e) Test the θ*-trans swept volume* corresponding to the selected collision free θ *swept volume.* The swept volume is collision free.

(f) Place the full range γ*-trans swept volume* corresponding to the selected θ*-trans swept volume* on the contour. The γ*-trans swept volume* collides with obstacles.

(g) Search the γ*-trans swept volumes* at lower levels in the γ*-trans swept volume* binary tree. The selected γ*-trans swept volume* collides with obstacles.

(h) Select another γ*-trans swept volume.* This γ*-trans swept volume* also collides with obstacles.

(i) Search the γ*-trans swept volumes* one level down in the γ*-trans swept volume* binary tree. The selected γ*-trans swept volume* collides with obstacles.

(j) Select another γ*-trans swept volume.* The selected γ*-trans swept volume* is collision free.

(k) Select another γ*-trans swept volume.* The selected γ*-trans swept volume* collides with obstacles.

(l) Select another γ*-trans swept volume.* The selected γ*-trans swept volume* is collision free. Up to now, there are two possible paths found in (i) and (l) respectively.

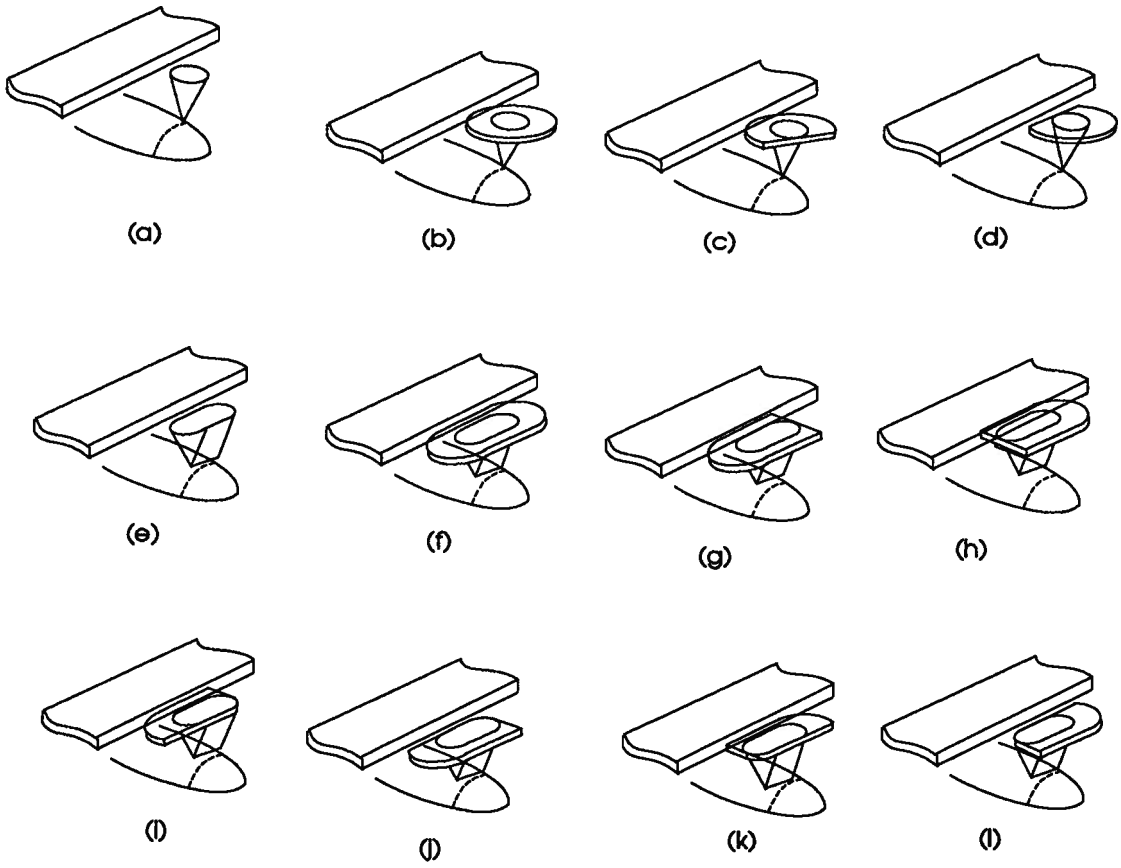

Figure 5.6: Procedure for searching for collision free paths for butchering.

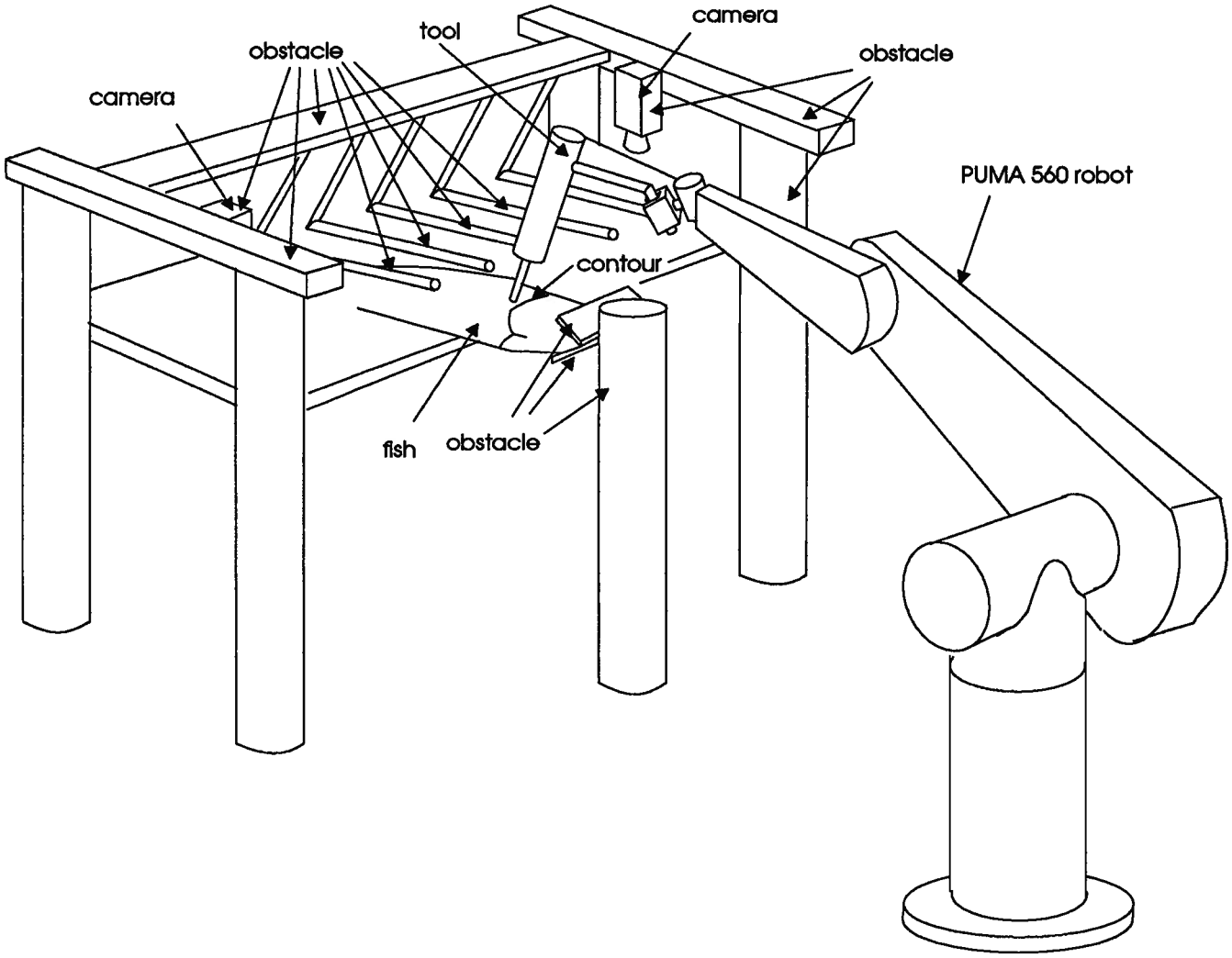

Figure 5.7: The automatic fish butchering workcell

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis a new trajectory planning algorithm, the Midpoint Algorithm, has been developed. The algorithm plans smooth trajectories for a robot tool moving along specified curves subject to application constraints. Geometric constraints (allowable orientation and position of tool) and motion constraints (joint limits, robot configuration change avoidance and collision avoidance) are also integrated into the algorithm. The trajectory is planned by searching for path segments between successive curve points, linking valid path segments to generate a path along the curve and selecting the optimal configurations inside the path to specify the final trajectory.

The configuration space of a robot tool forms a critical component of the algorithm. The configuration of robot tool has three degrees of freedom, α, β and γ. This configuration gives a general description of robot tool, which includes the tool axis derived from optimal orientation within an allowable tolerance and the tool itself rotating about the tool axis. The concepts of *Entire C-space* and *Local C-space* were introduced as a convenient representation of the tool configuration for planning purposes. Since the *Local C-space* only covers useful parts of the *Entire C-space*, a condensed data structure is used to represent the configuration space with high resolution. The *Local C-space* is the combination of the θ *region* and the γ *range*, which correspond to the constraints of allowable tool orientation and robot inverse kinematics respectively. Different *Local C-spaces* have

different shapes, sizes and positions of the corresponding θ *regions*. A mathematical expression for *Local C-space* has been derived, and given the optimal orientation (α_o, β_o) of the *Local C-space* and the maximum allowable orientation θ_o, the corresponding θ *region* can be calculated.

The trajectory planning is initiated using a Point Check procedure to search the *Local C-space* at the first curve point and initializes a *path list* of potential paths. A Translation and Rotation Check identifies collision free path segments by searching overlapping θ *regions* at successive points, shrinking the overlapping regions in order to find collision free regions. If no collision free γ ranges are found even though the minimum θ region has been tested, a midpoint is added if the step size is greater than the *minimum step size*, while the current path cannot be extended if the step size is less than the *minimum step size* .

Corresponding to the θ *regions* and γ *ranges*, swept volumes of the tool are generated and used to detect collision between the tool and the environment. Since only swept volumes which have different shapes are loaded into memory, the system memory requirements are reduced and exploration of high resolution configuration space is possible.

To address the problem of how to guarantee that the track of the tool tip matches the curve, a method of automatically varying the step size between curve points was developed. The step size is examined to determine: (1) if the straight line connecting successive curve point can approximate the shape of this curve segment, (2) if the optimal orientation at the points are close enough, and (3) if the environment between the two points is simple enough. After adjusting the step size, a minimum number of curve points are used to search for the path, which results in fast path planning. The small step size used in complex situations leads to safer paths. A polyline models the track of tool tip, which ensure that the tool does not collide with the workpieces.

An A^* search optimizes the path planning for multiple paths. Since the planner searches configurations along the entire curve, robot configuration changes can be avoided, which provides smooth motion of the tool. The planner also allows the tool to translate and rotate simultaneously without collision, which provides for a smooth motion of tool.

An off-line simulation program, TRAJPLAN, based on the Midpoint Algorithm has been developed. After the user inputs the application specifications, selects the curve to be processed, and identifies the environmental objects, TRAJPLAN automatically plans the trajectory of robot tool and generates a set of sequential joint angles corresponding to a particular robot. Currently the system supports the PUMA 560 and CRS A460 robots.

The Midpoint Algorithm has been demonstrated in software simulation of fish butchering and robotic welding. The algorithm is suitable to a variety of robot applications which require collision free motion subject to process constraints. By changing the accuracy levels of the parameters, the user specifies the precision of the trajectory or the kind of trajectory preferred in order to meet the different requirements of robot applications.

6.2 Recommendations for Future Work

This approach only considers the collision free motion of the robot tool, but not that of robot manipulator. Also dynamic constraints are not considered. An immediate extension to this algorithm includes the consideration of the entire robot and the inclusion of dynamic constraints in the planning process. Also more extensive testing on industrial applications needs to be done.

TRAJPLAN is a general algorithm for CAD environments which provide functions for building solids, moving solids and checking for interference between solids. To date TRAJPLAN has only been implemented for use with AutoCAD.

TRAJPLAN can also be used without a graphics interface after creating the data structures for storing solids and the interference check tool. Most of the time in path planning is spent on the interference check by AutoCAD and the display of graphics on the screen. Without the graphics interface, TRAJPLAN can plan the trajectory much faster and it may be possible to use it as a real-time software. There are three steps to build such a system:

1. Use sensors to obtain the information of environment and the curves to be processed.

2. Build a data structure to hold the information of the robot, tool and environment.

3. Create an interference check function using the above data structure.

Appendix A

Data Structure

Some data structures used in TRAJPLAN are described in this section.

1. The structures about curve and track,

```
Config = record
        alpha, beta, gamma : double ;
        end;

Point = record
        x, y, z : double ;
        end;

CurvePoint = record
        num : double ;
        curve_point : Point ;
        track_point : Point ;
        optimal : Config
        end;

Curve = record
        data : CurvePoint ;
```

```
        ↑ next : Curve
        end;
```

Here is the descriptions of above structures:

alpha, beta, gamma	the angle of tool configuration.
num	the number of curve point.
optimal	the optimal orientation of robot tool at the point.
curve_point	point in the curve.
track_point	point in the track of robot tool tip.
x, y, z	the coordinates of point.

2. The structures used to represent region and range in *Local C-space* and the corresponding swept volumes,

```
Region = record
        boundary : array [0..20][0..2] of double;
        ↑ next : Region
        end;

Thnode = record
        i, j, color : integer ;
        region : Region ;
        id : ap_Objid ;
        ↑ fname : char ;
        end;
```

```
Range = record
        MinG, MaxG : double ;
        end;

Gnode = record
        i, j, color : integer ;
        range : Range ;
        id : ap_Objid ;
        ↑ fname : char ;
        end;
```

Here is the descriptions of above structures:

boundary	the boundary of θ region calculated as discussed in Chapter 3.
MinG, MaxG	the boundary of γ range.
i, j	the index of a Gnode in binary tree.
color	the color of the node.
region	the θ region in Thnode.
range	the corresponding γ range of the node.
id	the swept volume object ID used for moving the swept volume and checking the interference.
fname	the swept volume file name used for loading the swept volume into memory.

3. The structures about robot inverse kinematic solution,

```
InvKinSol = record
        angle : array [0..6] of double;
        end;

InvKinSols = record
        config : Config ;
        solution : array [0..8] of InvKinSol;
        valid_num : double ;
        end;
```

Here is the descriptions of above structures:

angle	the array storing six robot joint angles.
config	the robot tool configuration.
solution	the array storing eight robot inverse kinematics solutions based on one tool orientation.
valid_num	the number of valid robot inverse kinematics solutions.

4. The structures about *path list*,

```
Part = record
        point_num : double ;
        point : CurvePoint ;
        region : Region ;
        range : Range ;
        ↑solutions : InvKinSols ;
```

```
        end;

Path = record
        data : Part ;
        ↑next : Path ;
        end;

PathList = record
        ↑ path : Path ;
        PathCost : double ;
        ↑ next : PathList ;
        end;
```

Here is the descriptions of above structures:

point_num	the number of the track point.
point	the information of track point.
region	the θ region of the *free-cell*$_{i,i+1}$.
range	the γ range of the *free-cell*$_{i,i+1}$.
solutions	the inverse kinematics solutions.
data	the information of a region.
path	the list of *free-cell*$_{i,i+1}$.
PathCost	the cost of the path.

5. The structure of robot tool trajectory composed of two parts: the configuration and position of tool, and the corresponding robot inverse kinematic solutions.

```
Trajectory = record
        point_num : double ;
        point : Point ;
        select : Config ;
        solution : InvKinSol ;
        ↑ next : Trajectory ;
        end;
```

Here is the descriptions of Trajectory structures:

point_num	the number of the curve point.
point	the track point.
optimal	the optimal orientation at the point.
select	the selected configuration.
solution	the robot inverse kinematics solution.

Appendix B

TRAJPLAN

The pseudo-code of the main procedure in TRAJPLAN is listed as follow:

```
var
    addmid, finish : integer;
    obstacleID : ap_objid;
    Cspace1, Cspace2 : Part;
program  Trajplan()
        { main procedure for trajectory planning }
var
    ↑ curve : Curve;
    ↑ paths : PathList;
    ↑ point, ↑ last_point : Curve;
    ↑ trajectory : Trajectory;
begin
    addmid := 0; finish := 0;
    Specification();
    curve := SelectCurve();
    obstacleID := SelectEnvironment();
    LoadSweptVolume();
    paths := PointCheck(point);
```

```
    if paths = NULL then exit; {No path found}
    while (finish not 1)or (finish not -1) begin
        last_point := NULL;
        last_point := SearchLastPoint(paths↑path↑data.point_num, curve);
        if last_point = NULL then exit;
        else if last_point↑next = NULL then
            finish := 1; {trajectory planing finish.}
        else begin
            paths := CheckLoop(paths, last_point, last_point↑next);
            if addmid = 0 then begin
                paths↑PathCost := PathCost(paths↑path);
                paths := SortPathList(paths);
                finish := 0;
            end
        end
    end
    if finish = -1 then exit; {No path found}
    trajectory := OptimizeTrajectory(paths↑path);
end

function CheckLoop( ↑paths:PathList, ↑last_point:Curve, ↑next_point:Curve) : PathList;
    { Check loop is applied to successive points to search the path segment between
    them. The Overlap Check and Inverse Kinematics Check are included in
    TranslateRotateCheck.}
var
```

```
    retval : integer;
begin
    addmid := 0;
    Cspace1 = BuildCspace( last_point↑data.optimal);
    Cspace2 = BuildCspace( next_point↑data.optimal);
    if not CurvatureCheck(last_point↑data, next_point↑data) then begin
        addmid := 1;
        curve := AddMidpiont(curve, last_point, next_point);
        return paths;
    end
    retval := TranslateRotateCheck(paths, last_point↑data, next_point↑data)
    if retval = 0 then begin { fres-part do not overlap.}
        addmid := 1;
        curve := AddMidpiont(curve, last_point, next_point);
        return paths;
    end
    else if retval = -1 then begin {no path segment found.}
        if PointCheck = NULL then begin
            finish := -1;
            exit;
        end
        else begin
            paths := DeletePath(paths);
            return paths;
    end
```

```
    else begin {the minimum cost path is extended}
        addmid := 0;
        paths↑path↑data.point_num := next_point↑data.num;
        return paths;
    end
end

function TranslateRotateCheck( ↑paths:PathList, last:CurvePoint, next:CurvePoint) : integer;
    { This function searches the free-region between last and next points.}
var
    length : double;
    shrink_center : Config;
    ↑ ov_region, ↑ free_region, ↑ ov_free_region : Region;
    ↑ free_range : Range;
    ↑ free_part : Part;
    ↑ solutions : InvKinSols;
begin
    length := Distance(last.data.track_point, next.data.track_point);
    if length < minimun_step_size then return -1;{avoid infinitive adding midpoint.}
    ov_region := OverlapCheck(Cspace1.region, Cspace2.region);
    if ov_region = NULL then return 0
    free_region := ShrinkOvRegion(ov_region, length, last, next);
    if free_region = NULL then return -1;
    free_range := FreeRange(free_region, length, last, next);
    if free_range = NULL then
```

```
        free_region := SmallRegion(free_region);
    if free_region = NULL then finish := 0; return -1;
    ov_free_region := OverlapCheck(paths↑path↑data.region, free_region);
    curve := AddMidpiont(curve, last_point, next_point);
    solutions := InverseKinematicCheck(ov_free_region, last_point↑next);
    if solutions = NULL then return -1;
    paths↑path↑data.region.solutions :=
                InverseKinematicCheck(ov_free_region, last_point↑next);
    if solutions = NULL then return -1;
    {If the inverse kinematic solution available, add midpoint region between last and next points.}
    paths↑path:= AddPart(ov_trans_free_region, last_point↑next);
    paths↑path↑data.region.solutions := InverseKinematicCheck(trans_free_region, next_point);
    if solutions = NULL then return -1;
    paths↑path:= AddPart(trans_free_region, next_point); {add region at next point.}
    return 1;;
end


{ The following primitive routines are needed to perform above procedures}
procedure Specification();
        { Input the specification of robot application. }
function SelectCurve() : Curve;
        { Read the curve point information of the selected curve and reconstruct curve.}
function SelectEnvironment() : ap_objid;;
        { Union the selected environment objects and return the objectID.}
procedure LoadSweptVolume();
```

{ Load the swept volumes into memory. }

function SearchFirstPoint(point) : RegionList;

{ Search white part in the *Local C-space* at the first point and store the parts into *path list.* }

function SortPathList(paths) : PathList

{ Sort the paths in the *path list* according to the PathCost and put the minimum cost path at the front of the list. }

function SearchLastPoint(path) : Curve;

{ Search the last point to be traveled by current path.}

function BuildCspace(optimal) : Part;

{ Calculate the θ *region* based on optimal orientation (α_o,β_o) and θ_o.}

function DeletePath(paths) : PathList;

{ Delete the top path from the *path list* and return the *path list.*}

function ShrinkOvRegion(ov_region, length, last, next) : Region;

{ Shrink the overlapping θ *region* to search the region of *free-cell*$_{i,i+1}$ and return the region}

function Distance(point1, point2) : double;

{ Calculate the distance between two track points and return the distance. }

function AddPart(part, path) : Path;

{ Add free part into path and return the path. }

function FreeRange(free_region, length, last, next) : Range;

{ Search collision free γ range of free_region and return the range. }

function BuildPart(free_region, free_range) : Part;

{ Build free_part based on free_region and free_range and return the part. }

function AddMidPoint(point, next_point) : Curve;

```
        { Add midpoint between last_point and next_point and return the curve. }
function CurvatureCheck( last_point, next_point ) : boolean;
        { Return True if two points can pass curvature check.}
function OverlapCheck(region1, region2 ) : Region;
        { Check the overlap between two regions and return the overlapping region. }
function PathCost(path) : double;
        { Calculate the PathCost of a path.}
function InverseKinematicCheck(region, point) : ↑Solutions;
        { Compute the robot inverse kinematics solutions. There are 24 configurations
        in the region to be selected and 8 inverse kinematics solutions can be obtained
        from each configuration. }
function OptimizeTrajectory(point) : Trajectory;
        { Use cost function to optimize the trajectory. }
```

Appendix C

Inverse Kinematic Modules for PUMA 560 and CRS A460

With reference to [Wolo 91] and Figure (C.1), the six Homogeneous transformation matrices of PUMA 560 robot are as follows:

$$H_0^1 = \begin{pmatrix} cos\theta_1 & -sin\theta_1 & 0 & 0 \\ sin\theta_1 & cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.1}$$

$$H_1^2 = \begin{pmatrix} cos\theta_2 & -sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -sin\theta_2 & -cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.2}$$

$$H_2^3 = \begin{pmatrix} cos\theta_3 & -sin\theta_3 & 0 & e \\ sin\theta_3 & cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & g \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.3}$$

$$H_3^4 = \begin{pmatrix} cos\theta_4 & -sin\theta_4 & 0 & 0 \\ 0 & 0 & 1 & f \\ -sin\theta_4 & -cos\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.4}$$

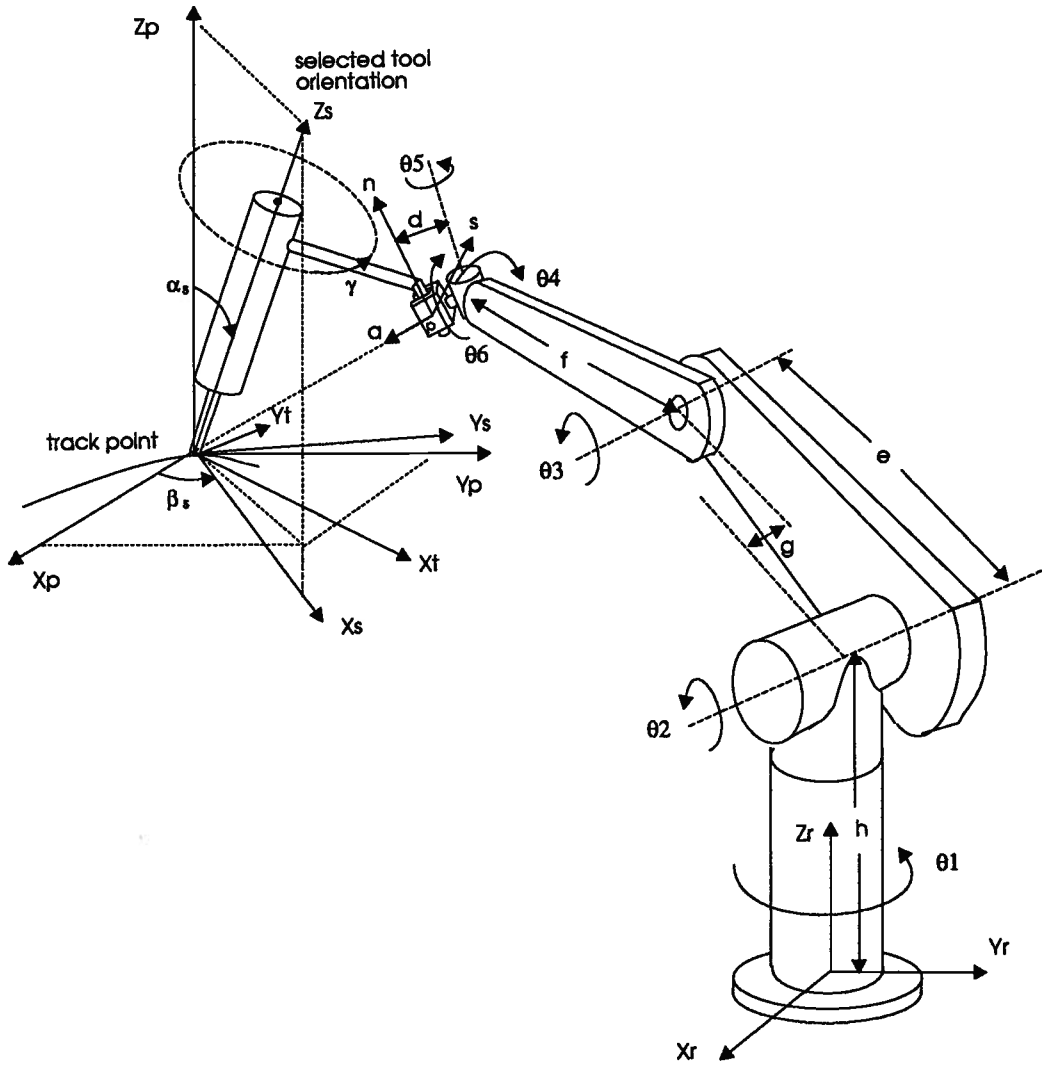

Figure C.1: The PUMA 560 robot and robot tool

$$H_4^5 = \begin{pmatrix} cos\theta_5 & -sin\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -sin\theta_5 & -cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.5}$$

$$H_5^6 = \begin{pmatrix} cos\theta_6 & -sin\theta_6 & 0 & 0 \\ 0 & 0 & -1 & -d \\ sin\theta_6 & cos\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{C.6}$$

The only difference between CRS A460 robot and PUMA 560 robot is the different values of parameters: d, e, f, g, and h. The Homogeneous transformation matrices (C.1), (C.2), (C.3), (C.4), (C.5), and (C.6) can also be used to describe CRS A460 robot. The computation subsequently discussed is for both the PUMA 560 robot and the CRS A460 robot.

Equation (4.7)

$$H_{r,e} = H_{r,p}H_{p,s}H_{s,t}H_{t,e}$$

is used in inverse kinematics computation. The $H_{r,e}$ can be rewritten as H_6^0 and is calculated as

$$H_6^0 = (H_5^6)^{-1}(H_4^5)^{-1}(H_3^4)^{-1}(H_2^3)^{-1}(H_1^2)^{-1}(H_0^1)^{-1} \tag{C.7}$$

From Equation (C.1) (C.2) (C.3) (C.4) (C.5) and (C.6), we know that Equation (C.7) only has variables: θ_1, θ_2, θ_3, θ_4, θ_5, θ_6, d, e, f, g, and h. The PUMA 560 geometry parameters, d, e, f, g, and h are known. The right hand side of Equation 4.7 represents the Homogeneous transformations from robot base to seam point and then end-effector. The End Effector Frame F_e $(\vec{a}, \vec{n}, \vec{s})$ and end-effector position p are attained, as shown in Figure C.1. The desired wrist position (P_{xw},P_{yw},P_{zw}) used in later computation is

calculated as

$$\begin{pmatrix} P_{xw} \\ P_{yw} \\ P_{zw} \end{pmatrix} = \begin{pmatrix} P_x - da_x \\ P_y - da_y \\ P_z - da_z \end{pmatrix} \tag{C.8}$$

The calculations for the robot joint angles are as follows [Wolo 91],

$$\theta_1 = Atan2\left[\frac{-P_{xw}}{P_{yw}}\right] + Atan2\left[\frac{\pm\sqrt{P_{xw}^2 + P_{yw}^2 - g^2}}{g}\right] \tag{C.9}$$

$$\theta_3 = Atan2\left[\frac{e^2 + f^2 + g^2 - P_{xw}^2 - P_{yw}^2 - (P_{zw} - h)^2}{\pm\sqrt{4e^2f^2 - [e^2 + f^2 + g^2 - P_{xw}^2 - P_{yw}^2 - (P_{zw} - h)^2]^2}}\right] \tag{C.10}$$

$$\theta_2 = Atan2\left[\frac{-(P_{xw}cos\theta_1 + P_{yw}sin\theta_1)fcos\theta_3 - (P_{zw} - h)(e - fsin\theta_3)}{(P_{xw}cos\theta_1 + P_{yw}sin\theta_1)(e - fsin\theta_3) - (P_{zw} - h)fcos\theta_3}\right] \tag{C.11}$$

$$\theta_4 = Atan2\left[\frac{a_xsin\theta_1 - a_ycos\theta_1}{a_xcos\theta_1cos(\theta_2 + \theta_3) + a_ysin\theta_1cos(\theta_2 + \theta_3) - a_zsin(\theta_2 + \theta_3)}\right] \tag{C.12}$$

$$and - \theta_5 \pm 180^o$$

$$\theta_5 = Atan2\left[\frac{\sqrt{(a_xsin\theta_1 + a_ycos\theta_1)^2 + (a_xcos\theta_1cos(\theta_2 + \theta_3) + a_ysin\theta_1cos(\theta_2 + \theta_3) - a_zsin(\theta_2 + \theta_3))^2}}{-a_xcos\theta_1sin(\theta_2 + \theta_3) - a_ysin\theta_1sin(\theta_2 + \theta_3) - a_zcos(\theta_2 + \theta_3)}\right] \tag{C.13}$$

$$and - \theta_5$$

$$\theta_6 = Atan2\left[\frac{-s_xcos\theta_1sin(\theta_2 + \theta_3) - s_ysin\theta_1sin(\theta_2 + \theta_3) - s_zcos(\theta_2 + \theta_3)}{n_xcos\theta_1sin(\theta_2 + \theta_3) + n_ysin\theta_1sin(\theta_2 + \theta_3) + n_zcos(\theta_2 + \theta_3)}\right] \tag{C.14}$$

$$and - \theta_6 \pm 180^o$$

Bibliography

[Ange 88] Angeles, J., Rojas, A. and Lopez-Cajun, C.S., *"Trajectory Planning in Robotics Continuous-Path Applications"*, IEEE Transactions on Robotics and Automation, Vol.4, No.4, 1988. pp.380-385.

[Barr 89] Barry, P. and Goldman, R., *"A Recursive Evaluation Algorithm for a Class of Catmull-Rom Splines"*, SIGGRAPH 89, pp.199-204.

[Bars 80] Barsky, B.A. and Greenberg D.P. *"Determining a Set of B-spline Control Vertices to Generate an Interpolating Surface"*, Computer Graphics and Image Processing, No.14, 1980, pp.203-226.

[Bart 87] Bartels, R., Beatty J. and Barsky, B.A., *"An Introducetion to Spline for Use in Computer Graphics and Geometric Modeling"*, by Morgan Kaufmann, Los Altos, CA, 1987.

[Boll 71] Bollinger, J.G. and Harrison, H.L., *"Automatied Welding Using Spacial Seam Tracing"*, Welding Journal, November 1971. pp.787-792.

[Broo 83(a)] Brooks, R.A., *"Solving the Find-Path Problem by Good Representation of Free Space"*, IEEE Transactions on Systems, Man and Cybernetics, SMC Vol.13, No.3, 1983, pp.190-197.

[Broo 83(b)] Brooks, R.A. and Lozano-Pérez, T., *"A Subdivision Algorithm in Configuration Space for Findpath with Rotation"*, Proceedings of the 8th International Conference on Artificial Intelligence, Karlsruhe, FRG, 1983, pp.799-806.

[Buch 89] Buchal, R.O., Cherchas, D.B., Sassani, F. and Duncan, J.P., *"Simulated Off-Line Programming of Welding Robots"*, International Journal of Robotics Research. Vol.8, No.3, 1989, pp.31-43.

[Buck 89] Buckley, S.J., *"Fast Motion Planning for Multiple Moving Robots"*, Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, 1989, pp.322-326.

[Cann 88] Canny, J.F., *"The Complexity of Robot Motion Planning"*, by The MIT Press, 1988.

[Cann 90] Canny, J.F. and Lin, M.C., *"An Opportunistic Global Path Planner"*, Proceedings of the IEEE International Conference on Robotics and Automation, 1990, pp.1554-1559.

[Catm 74] Catmul, E. and Rom, R., *"A Class of Local Interpolating Splines"*, in Barnhill, R.E. and Riesenfeld, R.F. eds., Computer Aided Geometric Design, Academic Press, San Francicisco, 1974. pp.317-326.

[Chaz 87] Chazelle, B., *"Approximation and Decomposition of Shapes"*, in Schwartz, J.T. and Yap, C.K., Algorithmic and Geometric Aspects of Robotics, Lawrence Erlbaum Associates, Hillsdale,NJ, 1987. pp.145-185

[Fole 91] Foley, J.D., Van dam, A., Feiner, S.K. and Hughes, J.F., *"Computer Graphics"*, by Addison-Wesley Publishing Company, 1991.

[Gama 93] Gamage, L.D.K.B., *"A Model-Based Approach to Complex Contour Generation for Process Automation Using Computer Vision* Ph.D Dissertation, Department of Mechanical Engineering, University of British Columbia, 1993.

[Gupt 90] Gupta, K.K., *"Fast Collision Avoidance for Manipulator Arms: A Sequential Search Strategy"*, IEEE Transactions on Robotics and Automation, Vol.6, No.5, 1990. pp.522-532.

[Kamb 86] Kambhampati, S. and Davis, L.S., *"Multiresolution path Planning for Mobile Robots"*, IEEE Journal of Robotics and Automation, vol. RA-2, No.3. 1986. pp.135-145.

[Khat 86] Khatib, O., *"Real-Time Obstacle Avoidance for Manipulators and Mobile Robots"*, International Journal of Robotics Research. Vol.5, No.1, 1986. pp.27-41.

[Khos 85] Khosla, P.K., Neuman, C.P. and Prinz, F.B., *"An Algorithm for Seam Tracking Applications"*, International Journal of Robotics Research. Vol.4, No.1. 1985. pp.27-41.

[Lato 91] Latombe, J., *"Robot Montion Planning"*, by Kluwer Acakemic Publishers, 1991.

[Loza 79] Lozano-Pérez, T. and Wesley, M.A., *"An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles"*, Communications of the ACM, Vol.22, No.10, 1979, pp.560-570.

[Loza 81] Lozano-Pérez, T., *"Automatic Planning of Manipulator Transfer Movements"*, IEEE Transactions on Systems, Man and Cybernetics. Vol.11, No.10, 1981, pp.681-698.

[Loza 83] Lozano-Pérez, T., *"Spatial Planning: A Configuration Space Approach"*, IEEE Trans. on Computer. vol. c-32. no.2 February, 1983, pp.108-119.

[Loza 87] Lozano-Pérez, T., *"A Simple Motion-Plannining Algorithm for General Robot Manipulators"*, IEEE Journal of Robotics and Automation, Vol. RA-3, No.3, June 1987, pp.224-237.

[Nils 69] Nilsson, N.J., *"A Mobile Automaton: An Application of Artificial Intelligence Techniques"*, Proceedings of the 1st International Joint Conference on Robotics and Automation, Washington D.C., 1969, pp.509-520.

[Odon 89] O'donnell, P.A. and Lozano-Pérez, T., *"Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators"*, Proceedings of the IEEE International Conference on Robotics and Automation, 1989, pp.484-489.

[Reif 85] Reif, J.H. and Sharir, M., *"Motion Planning in the Presence of Moving Obstacles"*, Proceedings of the 25th IEEE Symposum on Foundations of Computer Science, 1985, pp.144-154.

[Schw 83] Schwartz, J.T. and Sharir, M., *"On the Piano Movers' Problem:3. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers,"*, International Journal of Robotics Research. Vol.2, No.10, 1983, pp.97-140.

[Shaf 92] Shaffer, C.A. and Herb, G.M., *"A Real-Time Robot Arm Collision Avoidance System"*, IEEE Transactions on Robotics and Automation, Vol.8, No.2, 1992. pp.149-160.

[Shar 89] Sharir, M., *"Algorithmic Motion Planning in Robotics"*, IEEE Computer. March, 1989, pp.9-19.

[Sing 91] Singh, S.K. and Leu, M.C., *"Manipulator Motion Planning in the Presence of*

Obstacles and Dynamic Constraints", International Journal of Robotics Research. Vol.10, No.2, April 1991, pp.171-187.

[Taka 89] Takahashi, O. and Schilling, R.J., *"Motion Planning in a Plane Using Generalized Vorinoi Diagrams"*, IEEE Transactions on Robotics and Automation, Vol.5, No.2, April 1989. pp.143-150.

[Tomi 80] Tomizuka, M., Dornfeld, D. and Purcell, M., *"Application of Microcomputers to Automatic Weld Quality Control"*, Journal of Dynamic Systems, Measurement and Control. Vol.102, June 1980, pp.63-68.

[Tour 86] Tournassoud, P., *"A Strategy for Obstacle Avoidance and Its Application to Multi-Robot Systems"*, Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, 1986, pp.1224-1229.

[Udup 77] Udupa, S.M., *"Collision Detection and Avoidance in Computer Controlled Manipulators"*, Proc. 5th International Conferenec of Artificial Intelligence, 1977, pp.737-748.

[Warr 89] Warren, C.W., Danos, J.C. and Mooring, B.W., *"An Approach To Manipulator Path Planning"*, International Journal of Robotics Research. Vol.8, No.5, October 1989.

[Wang 92] Wang, D. and Hamam, Y., *"Optimal Trajectory Planning of Manipulators With Collision Detection and Avoidance"*, International Journal of Robotics Research. Vol.11, No.5, October 1992, pp.460-498.

[Wins 84] Winston, P.H., *"Artificial Intellengence"*, by Addison-Wesley Publishing Company, Inc. 1984.

[Wolo 91] Wolovich, W.A., *"Robotics: Basic Analysis and Design"*, by CBS College Publishing, 1991.

[Zhu 91] Zhu, D. and Latombe, J., *"New Heuristic Algoriths for Efficient Hierarchical Path Planning"*, International Journal of Robotics Research. Vol.7, No.1, February 1991, pp.9-20.