SURFACE MODELLING AND SURFACE FOLLOWING FOR ROBOTS EQUIPPED WITH RANGE SENSORS

This thesis is

presented to the

Department of Computer Science

for the degree of

Doctor of Philosophy

of the

University of Western Australia

By

Christopher John Pudney

September 1994

Abstract

The construction of surface models from sensor data is an important part of perceptive robotics. When the sensor data are obtained from fixed sensors, the problem of occlusion arises. To overcome occlusion, sensors may be mounted on a robot that moves the sensors over the surface. In this thesis the sensors are single-point range finders. The range finders provide a set of sensor points, that is, the surface points detected by the sensors. The sets of sensor points obtained during the robot's motion are used to construct a surface model. The surface model is used in turn in the computation of the robot's motion, so surface modelling is performed on-line, that is, the surface model is constructed incrementally from the sensor points as they are obtained.

A planar polyhedral surface model is used that is amenable to incremental surface modelling. The surface model consists of a set of model segments, where a neighbour relation allows model segments to share edges. Also sets of adjacent shared edges may form corner vertices. Techniques are presented for incrementally updating the surface model using sets of sensor points. Various model segment operations are employed to do this: model segments may be merged, fissures in model segment perimeters are filled, and shared edges and corner vertices may be formed. Details of these model segment operations are presented.

The robot's control point is moved over the surface model at a fixed distance. This keeps the sensors around the control point within sensing range of the surface, and keeps the control point from colliding with the surface. The remainder of the robot body is kept from colliding with the surface by using redundant degrees-of-freedom. The goal of surface modelling and surface following is to model as much of the surface as possible. The incomplete parts of the surface model (non-shared

edges) indicate where sections of surface that have not been exposed to the robot's sensors lie. The direction of the robot's motion is chosen such that the robot's control point is directed to non-shared edges, and then over the unexposed surface near the edge.

These techniques have been implemented and results are presented for a variety of simulated robots combined with real range sensor data.

Preface

Most of the research presented in this thesis has been published previously. Also there is much research into surface modelling and surface following for planar robots that has also been published by the author. This research on planar robots lead to the surface modelling and surface following techniques for robots operating in three-dimensional environments, presented in this thesis.

Early surface modelling techniques using two-dimensional sensor data have been published in the *Third Annual Conference on AI, Simulation and Planning in High Autonomy Systems* [83], and in greater detail in the *U.W.A. Department of Computer Science 1992 Annual Research Conference* [84]. These techniques were generalized to surface modelling using three-dimensional sensor data, and are presented in Chapter 2. From this chapter, details of the surface model, surface model updating, and examples of model segment operations have been published in the *First Australian and New Zealand Conference on Intelligent Information Systems* [86] and in greater detail in the *17th Annual Computer Science Conference* [87].

Early surface following techniques for a simple two-link arm appear in the *U.W.A. Department of Computer Science 1991 Annual Research Conference* [82]. These techniques were generalized to planar n-link manipulators, and this work has been published in the *Third Annual Conference on AI, Simulation and Planning in High Autonomy Systems* [83], and in greater detail in the *U.W.A. Department of Computer Science 1992 Annual Research Conference* [84]. These techniques were then generalized to robots operating in three-dimensional environments, and are presented in Chapter 3. From this chapter, details of the control point motion and use of redundancy have been published in the *U.W.A. Department of Computer Science 1993 Annual Research Conference* [85].

Acknowledgements

My supervisor, Prof. Robyn Owens introduced me to robotics when I was an undergraduate. Since then her guidance and advice has proven invaluable to my research. Her thorough reading of many drafts of my thesis and technical papers has significantly improved their content and presentation. Prof. Owens is an inspiration and I am eternally grateful to her.

The Robotics and Vision Research Group at U.W.A. has provided me with an ideal environment in which to develop the ideas presented in this thesis. I have gained much from many discussions with the RobVis group, and the seminars presented to and by the group.

I am indebted to Laurie McKeaig for his technical expertise in the construction of the sensors described in Chapter 4. I also acknowledge the Michigan State University's Pattern Recognition and Image Processing Laboratory for permission to use the range images presented in Chapter 4.

I thank the heads of the Department of Computer Science, Prof. J. Rohl, Prof. G. Roy and Prof. C. Tsang for allowing me to use the excellent facilities provided by the department. I also acknowledge the financial support provided by the University of Western Australia in the form of a Commonwealth Postgraduate Research Award.

The fellowship and assistance of the staff and students of the Department of Computer Science is highly prized. During my time at U.W.A. I have formed many friendships that will stand the test of time and distance.

Finally, my family and friends have always been there for me. I will not forget the patient support and encouragement they have provided, especially Sheree who has been beside me since the beginning. Thank you all for everything.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

If you close your eyes, and place your hand on the surface of an object then you can feel the shape of the surface that your hand is touching. As you move your hand along the surface, you receive more information about the surface. This allows you to construct a mental model of the surface from the tactile information received from your hand. From this model you can determine the directions in which to move your hand along the surface, so as to explore untouched regions of the surface. You can continue in this manner until you have explored as much of the surface as your hand can reach. The mental model that you construct provides information about the shape and position of the surface. This information may be useful in subsequent tasks that involve the surface.

The automation of the human process described above, forms the theme of this thesis. Automation is achieved with a sensor-equipped robot. The robot moves its sensors over the surface of an object. During this motion the information obtained from the sensors is used to construct a model of the surface. The surface model is used in turn to determine the directions in which the robot should move its sensors, so as to further explore the surface.

The goal of robotics is to automate tasks that are hazardous, tedious or difficult for humans to perform. To achieve this goal, robots must be able to operate autonomously, that is, with little or no requirement for human intervention. It has long been recognised that in order to achieve this autonomy, robots must be able to perceive their environments, that is, robotic systems must be equipped with

sensors, and must use the sensory information available from these sensors.

A common approach is to use sensory information to construct a model of the robot's environment. This can be achieved by constructing models of the surfaces in the robot's environment. This process is called *surface modelling* and the model constructed is called a *surface model*. There are many surface properties that can be modelled, and the properties modelled often depend on the types of sensory information that are available. In this thesis, the surface properties of interest are geometric properties, namely the shape and position of the surface. As most robotics is performed in three-dimensional environments, this thesis investigates surface modelling in three-dimensions.

Once a surface model has been constructed, it can subsequently be used in the execution of the robot's tasks. Surface models are of importance to many robotic tasks, for example:

Object recognition: In model-based object recognition[1], features observed in sensor data are matched with a database of solid models. In some cases, surfaces are the primitives used for matching [32,33,34,37,38,39,40,77,90]. As a result, surface descriptions must be extracted from sensor data, that is, surface modelling must be performed. Also the database models may be constructed automatically by presenting objects to a sensor system, and constructing surface models from the sensor data. Oshima and Shirai [77,90] present objects to a range finder, and construct surface descriptions (models) of the objects from the range data. This is called the learning phase. The learned descriptions form the database of models used in the recognition phase.

Path planning: Path planning often requires *a priori* knowledge of the shape and position of obstacles in the robot's environment. Surface models of the obstacles can provide this information. Once the obstacles' surfaces have been modelled, paths can be planned using a variety of techniques, including the construction of the configuration-space obstacles/free-space [7,17,36,46,48,65,78],

[1]Besl and Jain [12], Chin and Dyer [23], and Marshall and Martin [70] survey model-based object recognition.

the construction of potential fields [26,57,63,73,94,98], and various other approaches [18,44,47,52,89,72].

Manipulation: Knowledge of the shape and position of an object is important when it comes to manipulating the object. A surface model of the object can provide this information. The position information allows the robotic system to locate and approach the object, and the shape information allows the robot to form a grasp with which to grip and manipulate the object [69,74,79,95,96].

1.1 Review

This section reviews existing sensor-based three-dimensional surface modelling techniques. These techniques can be grouped into three categories:

- shape from monocular images,
- image segmentation, and
- the use of robot-mounted sensors.

1.1.1 Shape from Monocular Images

In principle it is impossible to obtain three-dimensional information from a monocular image. However, given knowledge of how an image is formed, monocular images can provide three-dimensional surface orientation information[2]. Surface orientation can then be used to infer the shape of a surface.

Horn [49] proposes techniques for inferring surface shape from shading information. A reflectance map can be constructed that provides a unique mapping from surface orientation to image intensity for specific lighting conditions. Given the intensity value of an image pixel, a contour in the reflectance map can be determined that specifies the orientation of the surface point corresponding to the pixel. To obtain the surface point's orientation precisely, second and third images of the

[2] Aggarwal and Chien [1] survey techniques for extracting 3-D information from 2-D images.

surface under different lighting conditions are obtained. The contours from the reflectance maps for these lighting conditions are used to resolve the surface point's orientation. The use of multiple images and lighting conditions is called photometric stereo [100]. The reflectance maps are implemented as a look up table. Images of a calibration object of known surface shape are obtained under various lighting conditions. The intensity values obtained for a particular surface point are then used to fill an entry in the look up table.

When a surface has repeated patterns or texture, its orientation can be obtained from the texture gradient of its image. The texture gradient is the change in density and size of texture elements (texels). An example of this is the apparent vanishing line produced by a perspective projection. The orientation of a plane can be obtained from the vanishing line. Kender [55] obtains the vanishing line using image lines known to be parallel, while Ohta *et al.* [75] use the ratio of the areas of pairs of texels. Ikeuchi [50] measures the distortion of a repeated known texture pattern to determine possible surface orientations at each texel. The possible orientations are then reduced to a unique orientation at each texel by the iterative propagation of a smoothness constraint starting from known orientations of occluding boundaries points.

1.1.2 Image Segmentation

Surfaces may be extracted from an image by segmenting the image into regions that are a close fit to some surface[3]. Most segmentation approaches use region growing techniques with surface fitting criteria as the region growth constraints. These techniques can be applied to both range and intensity images. In the former case the extracted surfaces correspond to physical surfaces in the scene, while in the latter case they correspond to intensity surfaces, and as such are useful for image reconstruction.

Oshima and Shirai [76,77,90] group (range) pixels into small surface elements, and fit a least squares plane to each surface element. Approximately planar regions are then grown by merging adjacent surface elements that have similar least squares

[3]Besl and Jain [12] and Bolle and Vemuri [16] survey surface based image segmentation.

planes. Regions are then classified as being planar, curved or undefined according to various measures of fit. Curved regions are extended by merging adjacent curved or undefined regions that satisfy a smoothness criterion. Quadric surfaces are then fitted to the curved regions.

Faugeras and Hebert [33,34] define a measure $E(R)$ of the quality of the fit of a surface to a region R. Surfaces are either planes or quadrics. The initial regions are obtained from a triangulation of the (range) image points. Regions are grown by merging the pair of adjacent regions R_i and R_j that yield the minimum error $E(R_i \cup R_j)$ of all adjacent pairs. Region growing stops when the global error $\sum E(R_i)$ exceeds some threshold $E_{\max}$.

Besl [11] and Jain [13] segment range and intensity images using variable-order surfaces. The surfaces are totally ordered (in ascending order: planar, biquadratic, bicubic, biquartic) such that a higher order surface can represent a lower order surface but not *vice versa*. Pixels are labelled according to the signs of the mean and Gaussian curvatures. This labelling provides an initial coarse segmentation of the image that is refined by growing regions using variable-order surface fitting. Initially, planar surfaces are fitted to seed regions obtained from the labelled pixels. Regions are grown iteratively from the seed regions by incorporating neighbouring pixels that closely fit the approximating surface. As the regions grow the error of fit and other measures are monitored. If the error grows too large then the order of the fitted surface is increased.

1.1.3 Robot-Mounted Sensors

The approaches mentioned thus far use fixed sensors. A problem inherent with fixed sensors is that of *occlusion*: when part of the surface is obscured from the sensor's view. For some tasks this may not be a problem, for example, many object recognition techniques can succeed when the object is partially occluded. However, other tasks, such as path planning, may require that the entire object's surface be modelled. To achieve this, multiple views of the surface must be presented to the sensors. This can be achieved in a number of ways:

- Multiple fixed sensors can be located at strategic points in the robot's workspace [43,97]. However, occlusion can still occur for some surface shapes.
- The object can be mounted on a turntable [15,51] and/or manipulated by a robot arm [70]. However, for some tasks, such as path planning, this may be infeasible.
- Sensors can be mounted on a robot and moved about the surface. This is the approach investigated in this thesis.

Traditionally, robot-mounted sensors have been used to aid the navigation of robots: Dickmanns [27,28] developed a dynamic vision system for controlling a vehicle capable of driving autonomously on public roads. Elfes [29,30] produced a navigation and mapping system for a mobile robot equipped with sonar sensors. Flynn [41,42] fused data from a sonar range-finder and infra-red sensors mounted on a mobile robot to form a room boundary map for navigation indoors. Blake *et al.* [14] used the motion of image contours for the movement of a robot-mounted camera around curved obstacles. Cheung and Lumelsky [19,21,66] developed a whole-body sensitive skin made of infra-red sensors mounted on a flexible circuit board. The skin was mounted on a robot arm and used to provide sensory input to a continuous path planner for the arm. Espiau and Boulic [31] presented a path planner for a kinematically-redundant manipulator equipped with proximity sensors. Karlen *et al.* [53] also worked with a kinematically-redundant robot arm. The arm avoided obstacles reflexively using inputs from ultrasonic sensors fitted to the arm.

There is an increasing use of robot-mounted sensors for surface modelling and exploration. To date most of the robot-mounted sensors used for surface modelling have been tactile sensors. As a result, the sensors often measure the surface's material properties, such as compliance, texture and elasticity, as well as the surface's geometric properties. Only modelling of the surface's geometric properties is discussed in this thesis.

Ahmad and Lee [2,3] use a robot-mounted probe to track a planar contour of a surface. During motion of the probe, the robot's joint positions are recorded and

used to reconstruct the shape of the traced contour. This process provides a single contour of the surface. A number of such curves can be used to construct a contour map of the surface.

Dario and Buttazzo [25] measure surface shape using a robot-mounted tactile "finger". At each step of the finger's motion, the position of the finger and the tangent plane between the finger and the surface are recorded. The position and tangent plane are used to define a small planar rectangle. As the finger is swept across a surface, a set of overlapping rectangles is obtained that provides a description of the surface's shape.

Allen [4] constructs a Coons' patch [35] description of a surface. A robot-mounted tactile sensor traces a grid over a section of the surface. Cubic least squares polynomial curves are fitted to the tactile traces and used to provide the support for the Coons' patches. Allen [5] and Michelman [6] have also worked with a tactile multi-fingered hand. A series of enclosing grasps of an object is performed and a superquadric [8] fitted to the sensor data obtained from these grasps. Any planar surface sections identifiable from the superquadric surface are then examined by a series of one-finger proddings of the planar section. A least squares plane is fitted to these sensor data. An axis of rotation is identified on the object and then a series of two-finger encircling grips of the object is performed. This provides a series of planar surface contours.

Stansfield [91,92] classifies surface sections as being curved or planar. A robot-mounted tactile sensor is repositioned at a surface point until the maximum area of contact is obtained on the tactile array. The surface normal and position are then obtained from the sensor frame's orientation and position. The surface normal and position are measured in this manner at four local contact points on the surface, and the Euler angles of the sensor frame at each contact point are recorded. If the standard deviation of the angles is small then the surface section is classified as planar, otherwise it is considered to be curved. More recently, Stansfield has experimented with a multi-fingered tactile hand [93]. A section of surface is classified as planar, singly curved, or doubly curved by measuring the fingers' joint positions when the surface is grasped.

The techniques described above only extract information about part of the surface, not the entire surface. These techniques are sometimes called *exploratory procedures*. To obtain information about the entire surface, exploratory procedures need to be applied at a number of points on the surface. A high-level controller is required to coordinate the execution of exploratory procedures. The controller determines the location to which the robot should move its sensors in order to perform an exploratory procedure. Allen and Stansfield have implemented such high-level controllers.

The goal of Allen's robotic system is model-based object recognition. As well as surface descriptions, the database models also include features, such as holes and cavities. The exploratory procedure for building surface descriptions is described above. Allen also defines exploratory procedures for investigating holes and cavities. A fixed stereo vision system obtains an image of the object. Regions of interest on the object's surface are identified from the image, and examined by applying tactile exploratory procedures. The tactile and visual data are integrated into surface and feature descriptions that are matched with those in the database. This matching is used to instantiate a model consistent with the sensory information. The instantiated model is then verified by further tactile sensing to see if it is correct. This may lead to exploration of surfaces occluded from the vision system. Thus exploration proceeds by investigating regions of interest identified from visual data, and then as part of a process of verification of an instantiated model.

The goal of Stansfield's robotic system is the construction of a hierarchical model of an object. An object is described by its components (body and parts), and each component is described by a set of feature frames. There is a feature frame for each direction (left, right, front and top) from which a component is examined. The feature frames consist of slots that correspond to features to be extracted. Stansfield defines an exploratory procedure for extracting each type of feature. The exploratory procedure for classifying surface shape is described above. A fixed stereo vision system obtains an image of the object. The image is segmented so as to identify the object's body and part components. Some edges are also extracted. The left, right, front and top of each component is then explored with the tactile sensor.

Feature slots are filled by invoking the appropriate exploratory procedures. Thus exploration proceeds by examining each of the object's components from various directions, and extracting features of each component so as to fill slots in the object model.

1.2 Overview

In this thesis, sensor data are obtained using robot-mounted sensors. This section describes the robot and sensor arrangement, and presents an overview of the techniques used to construct three-dimensional surface models from the sensor data, and the techniques used to move the sensors using the robot.

As surface modelling is to be performed in three dimensions, the robot must be able to move its sensors in three dimensions. The robot may be a mobile robot or a manipulator arm, although in reality most mobile robots are restricted to motion in two dimensions. The robot's sensors are mounted around a point on the robot called the *control point*. The control point can be any point on the robot, although if the robot is a manipulator arm then for greatest dexterity the control point should be on the end-effector.

The sensors are range sensors[4]. A range sensor measures the distance to a surface along its sensing axis. Unlike tactile sensors, range sensors are non-contact devices, and so can be used without interfering with the surface. However, range sensors cannot measure a surface's physical properties. As a result, only geometric properties of the surface are modelled. To the best of my knowledge, the research presented in this thesis is the first application of robot-mounted range sensors to the task of three-dimensional surface modelling.

The range sensors have a relatively short sensing range[5]. As a result, the robot must move its sensors quite close to the surface in order to obtain sensor data. The robot's motion over the surface is called *surface following*. It is assumed that when surface following commences, the control point has been positioned close enough to the surface so that at least one range sensor can detect the surface. Unlike

[4]Coiffet [24] and Koenigsberg [61] survey range sensing technology.
[5]Long range sensors do exist, for example laser range finders.

Allen's and Stansfield's robotic systems, no *a priori* information about the surface is available for moving the sensors to the surface.

The range sensors are single-point range finders, that is, each sensor provides a distance reading to a single point on the surface. Thus the range data provided by a single operation of the robot's sensors are quite sparse. However, as the robot moves its sensors over the surface, the cumulative range data set can become quite dense. This is in contrast to most vision and range imaging systems that provide a dense image from a single operation of the sensors. Also the robot-mounted range sensor data are not ordered on some regular grid or array as in most vision and range imaging systems.

Given a range sensor's position $\mathbf{s}$, the direction of its sensing axis $\hat{\mathbf{a}}$, and its distance reading d (assuming it detects the surface) the coordinates of the surface point $\mathbf{p}$ detected by the sensor can be determined:

$$\mathbf{p} = \mathbf{s} + d\hat{\mathbf{a}}, \tag{1}$$

where $\hat{\mathbf{a}}$ is unit vector. Typically, $\mathbf{s}$ and $\hat{\mathbf{a}}$ are defined relative to a coordinate frame affixed to the robot, so $\mathbf{p}$ is transformed to its definition $\mathbf{p}'$ relative to a global coordinate frame. Since the sensors are mounted on the robot, this transformation involves the robot's kinematics [71]. The point $\mathbf{p}'$ is called a *sensor point*. The surface model is constructed using these sensor points.

Experiments were conducted into the construction of cheap single-point range finders from infra-red light emitting diodes. Unfortunately, the accuracy required could not be achieved using these components. As our budget did not extend to mounting several laser-range finders on a robot, a compromise was reached by using kinematic simulations of robots combined with real range data from laser range scanner images.

The control point is moved over the surface while its sensors collect data to be used for surface modelling. The surface model is constructed as this data is obtained. The rest of the robot body must also be equipped with sensors to prevent it from colliding with the surface. Thus the whole robot body can be covered with range sensors, in which case if part of the robot is near the surface then the sensors on this part of the robot will detect the surface. Their sensor data is then

incorporated into the surface model, and the surface model is used to prevent the robot from colliding with the surface. An alternative scheme is also considered, where only the control point is equipped with range sensors while the rest of the robot body is equipped with simpler proximity switches that are not used for surface modelling.

Surface modelling and surface following proceed in a stepwise manner:

1. The range sensors take surface readings. For each sensor that detects the surface, the sensor point is computed using (1).

2. The set of sensor points is used to update the surface model, and may then be discarded.

3. The updated surface model is used to determine the direction in which to move the control point along the surface, and the actuator changes that achieve this control point motion are computed.

4. The computed actuator changes are applied, causing the robot to move its sensors.

5. Steps 1,...,4 are repeated until as much of the surface as possible has been modelled.

Step 2 is the surface modelling step. Surface modelling is performed on-line, that is, the sensor points are used to update the surface model as soon as they are received. Thus the surface model is constructed incrementally from the sensor data. This is in contrast to the traditional approach of distinct sensor data collection and sensor data processing phases. With fixed sensor systems the sensor data are first obtained and then processed. Similarly, with robot-mounted sensor systems, an exploratory procedure is used to obtain sensor data, and then the sensor data are processed. The technique used here is to perform the collection and processing phases together. A sparse set of sensor data is obtained and processed, then the next sparse set of sensor data is obtained and processed, and so on.

Step 3 is the surface following step. It embodies the high-level controller that coordinates motion of the robot over the surface. The goal of surface modelling and

surface following is to model as much of the surface as possible. To achieve this, the robot's sensors must be exposed to as much of the surface as possible. From the surface model it can be determined where sections of unexposed surface lie. So surface following uses the surface model to determine where the robot should move its sensors. Surface following also uses the surface model to determine how the robot should move such that the sensors remain within sensing range of the surface, and the robot avoids collision with the surface. As the surface model provides surface following with so much information, the surface model must be as up-to-date as possible. This is the reason for performing surface modelling on-line.

1.3 Contributions

The task of three-dimensional surface modelling using fixed sensors suffers from the problem of occlusion. However, this can be overcome by using robot-mounted sensors. So far, only robot-mounted tactile sensors have been applied to the task of three-dimensional surface modelling. In this thesis, robot-mounted range sensors are used.

The goal is to model as much of the surface as possible. This is achieved through surface following, which moves the robot such that the sensors are exposed to as much of the surface as possible. To achieve this, surface following makes use of the surface model. As a result, surface modelling is performed on-line.

Thus the contributions of this thesis may be summarised as follows:

- Incremental surface modelling techniques that facilitate on-line surface modelling have been developed.

- Surface following techniques are described that use the surface model to move the robot over the surface so that as much of the surface as possible is modelled. During this motion, the sensors remain within sensing range of the surface, and the robot avoids colliding with the surface.

- The application of robot-mounted range sensors to the task of three-dimensional surface modelling is investigated.

1.4 Layout

The layout of this thesis is as follows: Chapter 2 presents surface modelling. The surface model is introduced, and the procedure for updating the surface model is presented. Surface modelling operations performed during the updating of the surface model are then described. Chapter 3 presents surface following. Means of determining the control point motion are described. Then the computation of the actuator changes that achieve the control point motion is given, and finally the algorithm used to choose the surface following direction is presented. In Chapter 4 single-point range finders are discussed, and results of surface modelling and surface following using real sensor data are presented and discussed. Chapter 5 concludes the thesis with a summary of the achievements and contributions of the thesis, and suggestions for future research. The bibliography and appendices are included at the end of the thesis.

Chapter 2

Surface Modelling

2.1 Introduction

This chapter presents the surface model and techniques used in incremental surface modelling. The surface model consists of a set of simple elements called model segments. A set of sensor points is used to update the surface model by creating a model segment for each of the sensor points. The creation of these model segments may result in the formation of gaps in the surface model. In an attempt to fill these gaps and reduce the complexity of the surface model, the following model segment operations are performed:

- Two model segments may be merged with each other to form a new model segment.
- Fissures in a model segment's perimeter may be filled.
- Shared edges may be formed between two model segments thus making the model segments neighbours.
- A corner vertex may be formed between successive shared edges of a model segment's perimeter.
- Shared edges may be unshared.
- Model segments may be deleted.

The layout of this chapter is as follows: In Section 2.2 the surface model is introduced. Section 2.3 describes how a set of sensor points is used to update the surface model. The remainder of the chapter is devoted to the presentation of model segment operations. In Section 2.4 methods for merging different types of model segments are presented. Section 2.5 describes means of filling fissures in the perimeters of model segments. In section 2.6 details of the formation of shared edges between model segments are given. Section 2.7 presents the methods used in the formation of corner vertices. In Section 2.8 the unsharing of shared edges is described. Section 2.9 gives the method used to delete model segments, and the chapter concludes with a summary in Section 2.10.

2.2 The Surface Model

As surface modelling is performed on-line, the surface model must be updated and the next robot motion computed as quickly as possible. In surface modelling and surface following, the bulk of computation performed with the surface model involves computing the distance to the surface model or parts of it. To simplify this distance computation, the surface model consists of a set of *model segments* having simple descriptions: either *points* or *planar polygons*. Each model segment represents a subset of the sensor points that have been processed so far, such that each sensor point is represented by at most one model segment.

A point type model segment's description is the point of best fit to the sensor points it represents, and a planar polygon type model segment's description is a planar polygon lying on the plane of best fit to the sensor points it represents. By "best fit" it is meant that the mean squared perpendicular distance of the sensor points from the point or plane is minimal. It is well know that the point of best fit to a set of points is given by the points' centroid:

$$(\bar{x}, \bar{y}, \bar{z}) = (\frac{\sum_{i=1}^{m} x_i}{m}, \frac{\sum_{i=1}^{m} y_i}{m}, \frac{\sum_{i=1}^{m} z_i}{m})$$

where m is the number of points, and (x_i, y_i, z_i), for $i = 1, \ldots, m$, are the points. Pearson [81] showed that the plane of best fit to a set of points is given by the

plane passing through the points' centroid, and having its normal aligned with the eigenvector associated with the smallest eigenvalue of the points' covariance matrix:

$$\mathbf{C} = \begin{bmatrix} \frac{\sum_{i=1}^{m} x_i^2}{m} - \bar{x}^2 & \frac{\sum_{i=1}^{m} x_i y_i}{m} - \bar{x}\bar{y} & \frac{\sum_{i=1}^{m} x_i z_i}{m} - \bar{x}\bar{z} \\ \frac{\sum_{i=1}^{m} x_i y_i}{m} - \bar{x}\bar{y} & \frac{\sum_{i=1}^{m} y_i^2}{m} - \bar{y}^2 & \frac{\sum_{i=1}^{m} y_i z_i}{m} - \bar{y}\bar{z} \\ \frac{\sum_{i=1}^{m} x_i z_i}{m} - \bar{x}\bar{z} & \frac{\sum_{i=1}^{m} y_i z_i}{m} - \bar{y}\bar{z} & \frac{\sum_{i=1}^{m} z_i^2}{m} - \bar{z}^2 \end{bmatrix}.$$

The eigenvectors of $\mathbf{C}$ are orthogonal to one another and aligned with the principal axes along which the points are spread about the centroid. If the points are orthogonally projected onto an axis that passes through the centroid and is aligned with an eigenvector then the associated eigenvalue gives the projected points' variance about the centroid. In order that the covariance matrix can be constructed, a model segment maintains the following *model segment data*:

$$\begin{array}{ccc} m, & & \\ \sum_{i=1}^{m} x_i, & \sum_{i=1}^{m} y_i, & \sum_{i=1}^{m} z_i, \\ \sum_{i=1}^{m} x_i y_i, & \sum_{i=1}^{m} x_i z_i, & \sum_{i=1}^{m} y_i z_i, \\ \sum_{i=1}^{m} x_i^2, & \sum_{i=1}^{m} y_i^2, & \sum_{i=1}^{m} z_i^2, \end{array}$$

where m is the number of sensor points represented by the model segment, and (x_i, y_i, z_i), for $i = 1, \ldots, m$, are the sensor points' coordinates relative to a global coordinate frame.

To prevent the sensor points from being spread too far from the point or plane of best fit, the sensor points' variance about the point or plane of best fit must be less than a preset threshold λ^2. For a point segment, the variance is:

$$s_{\text{pt}}^2 = \frac{\sum_{i=1}^{m} x_i^2 + y_i^2 + z_i^2}{m} - \bar{x}^2 - \bar{y}^2 - \bar{z}^2 = s_x^2 + s_y^2 + s_z^2,$$

and for a planar polygon segment, the variance s_{pp}^2 is given by the smallest eigenvalue. If $s_{\text{pt}}^2 < s_{\text{pp}}^2$ then the sensor points are better represented by a point segment, otherwise they are better represented by a planar polygon segment. So for a point segment $s_{\text{pt}}^2 < s_{\text{pp}}^2$ and for a planar polygon segment $s_{\text{pt}}^2 \geq s_{\text{pp}}^2$. In the case that the sensor points represented by the model segment are collinear, the model segment must be a point segment regardless of the values of s_{pt}^2 and s_{pp}^2. In this case, at least two of the eigenvalues are zero.

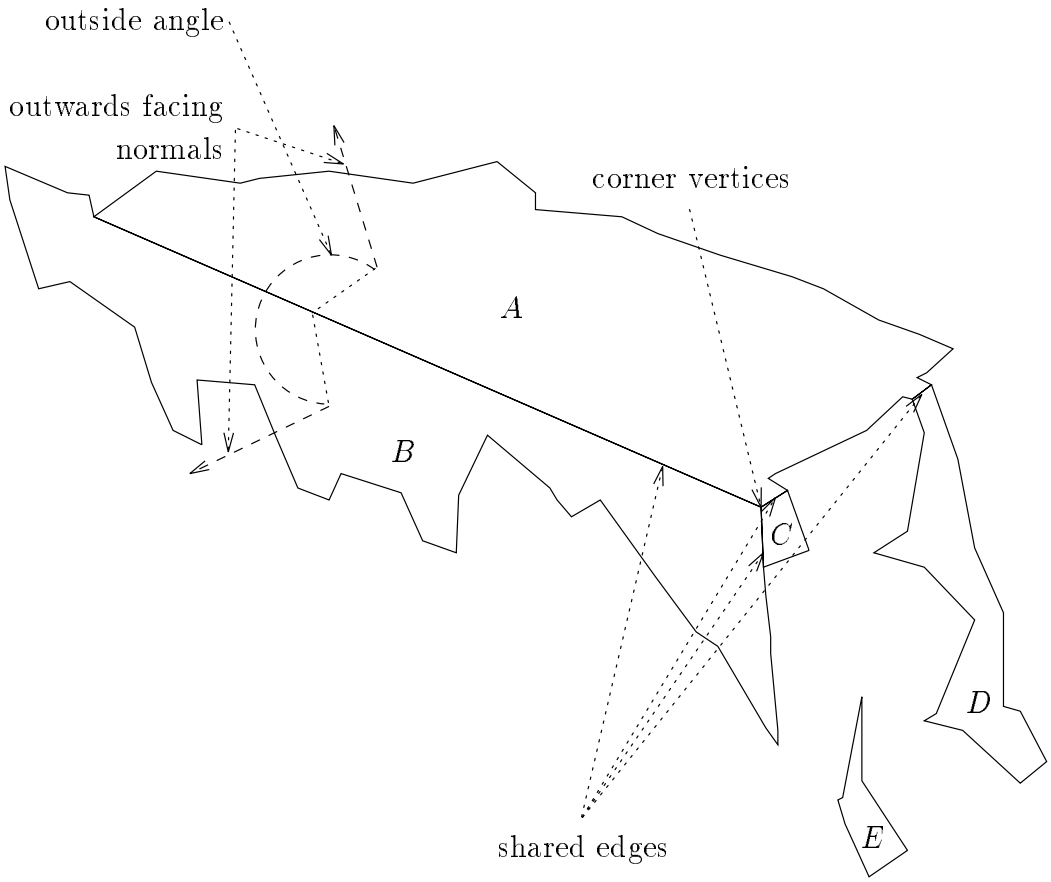


Figure 1: An example of a surface model.

A planar polygon segment's perimeter lies on the model segment's plane of best fit. The perimeter consists of a set of vertices connected by edges. The vertices are formed during model segment mergers and the formation of shared edges and corner vertices. Details of these operations will be given later in this chapter. To prevent the perimeter from crossing itself the edges may not cross each other. The planar polygon's plane is given an outside and an inside, and the plane's normal is made to face outside. The perimeter's direction is anticlockwise relative to the outwards facing normal.

Two planar polygon segments S_1 and S_2 may share edges. In such cases S_1 and S_2 are called *neighbours*. Point segments do not have edges and so cannot have neighbours. Figure 1 shows an example of a surface model having the following pairs of neighbours: (A, B), (A, C), (B, C) and (A, D). If S_1 and S_2 share an

edge then the perimeters of S_1 and S_2 maintain their own copy of the shared edge. These two copies of the shared edge form a pair of corresponding shared edges. If e_1 is an edge of S_1 shared with S_2, and e_2 is the corresponding edge of S_2 shared with S_1, then e_1 and e_2 lie along the line of intersection between the planes of S_1 and S_2, and overlap each other. Furthermore, the vertex at the start of e_1 in an anticlockwise direction coincides with the vertex at the end of e_2 in an anticlockwise direction, unless the vertices are corner vertices. Corner vertices will be described later. Similarly, the vertex at the end of e_1 in an anticlockwise direction coincides with the vertex at the start of e_2 in an anticlockwise direction, unless the vertices are corner vertices.

The *concavity* between a pair of neighbouring model segments is given by the concavity of the outside angle between the neighbours' planes at any edge they share, as shown in Figure 1. The outside angle is measured on the outside of the neighbours' planes. Although it is possible for two planar polygons to intersect with both types of concavity, neighbours are restricted to having one type of concavity. As will be explained in Section 2.6.1 this reduces the computational workload when forming shared edges. This concavity restriction is not unreasonable, as situations in which a pair of neighbours require both types of concavity are highly contrived, and even if both types of concavity are required then the surface model is only affected where the concavity cannot be represented.

A perimeter's direction is anticlockwise relative to the model segment's outwards facing normal. Therefore if S_1 and S_2 share edges then the anticlockwise direction along the edges S_1 shares with S_2 is opposite to the anticlockwise direction along the edges S_2 shares with S_1. If the concavity between S_1 and S_2 is convex then the anticlockwise direction along the edges S_1 shares with S_2 is $\mathbf{n}_1 \times \mathbf{n}_2$, and the anticlockwise direction along the edges S_2 shares with S_1 is $-(\mathbf{n}_1 \times \mathbf{n}_2) = \mathbf{n}_2 \times \mathbf{n}_1$, as shown in Figure 2(a). If the concavity between S_1 and S_2 is concave then the situation is reversed, that is, the anticlockwise direction along the edges S_1 shares with S_2 is $\mathbf{n}_2 \times \mathbf{n}_1$, and the anticlockwise direction along the edges S_2 shares with S_1 is $\mathbf{n}_1 \times \mathbf{n}_2$, as shown in Figure 2(b).

A *surface* is defined as a set of model segments that are neighbours of each other,

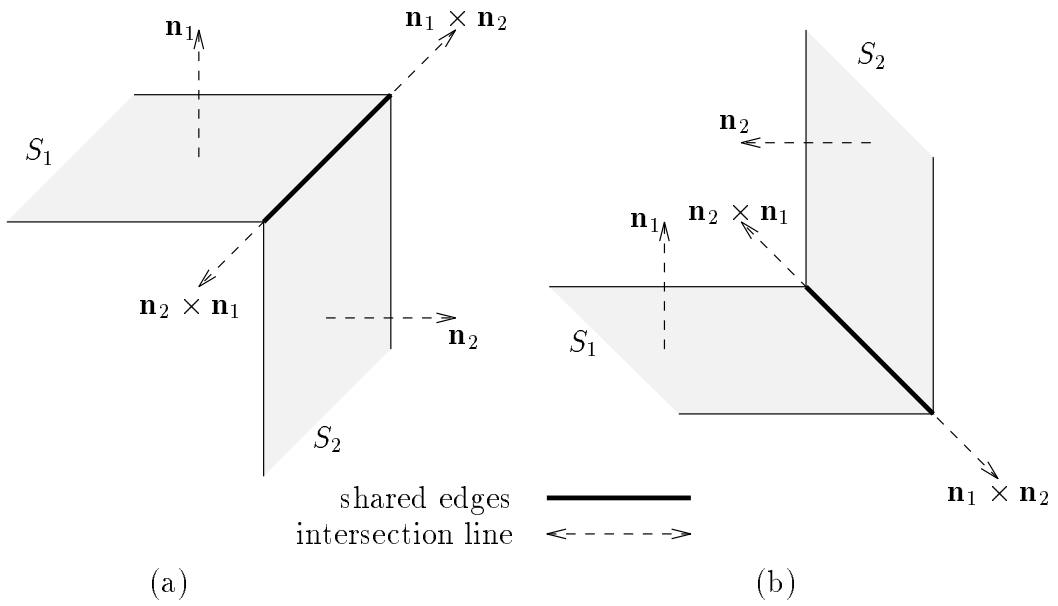


Figure 2: Anticlockwise directions along shared edges. (a) Convex. (b) Concave.

or neighbours of neighbours of each other, and so on. A model segment with no neighbours forms a surface on its own. In other words, a surface is a set of model segments connected by the neighbour relation. In Figure 1 there are two surfaces: $\{A, B, C, D\}$ and $\{E\}$.

A vertex that joins two edges shared with different neighbours is called a *corner vertex*. A corner vertex lies at the intersection of the two intersection lines along which the adjoining shared edges lie, as shown in Figure 1. Corner vertices are formed between sets of model segments $\{S_0, S_1, \ldots, S_n\}$, for $n \geq 2$, where S_i and $S_{(i+1) \bmod (n+1)}$ are neighbours. A corner vertex is formed for each S_i, $i = 0, \ldots, n$, at the point of intersection, if it exists, of the planes of S_h, S_i and S_j, where $h = (i+n) \bmod (n+1)$ and $k = (i+1) \bmod (n+1)$. As the planes of $S_0, S_1, \ldots, S_n$ may not intersect at a single point, these corner vertices are not guaranteed to be coincident.

To prevent the formation of infeasible perimeters, the ordering of shared edges is restricted (refer to Figure 3):

Local ordering: Suppose that S_1 and S_2 are neighbours, with $e_1, \ldots, e_n$ being the

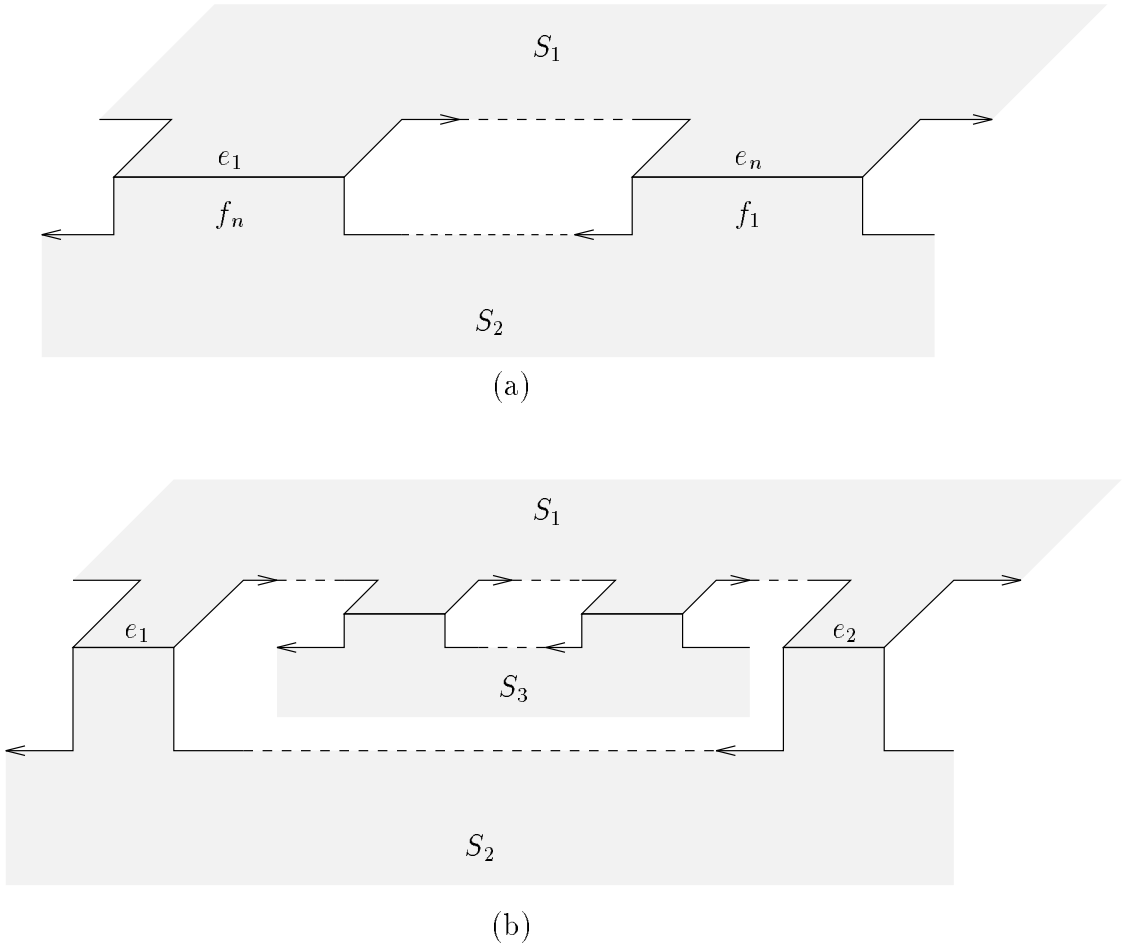


Figure 3: Shared edge ordering. Arrowheads indicate the anticlockwise direction along a perimeter. (a) Local ordering. (b) Global ordering.

edges of S_1 shared with S_2, and $f_1, \ldots, f_n$ being, respectively, the corresponding edges of S_2 shared with S_1. If $e_1, \ldots, e_n$ is the order in which the edges occur on the perimeter of S_1 then $f_n, \ldots, f_1$ is the order in which the edges must occur on the perimeter of S_2.

Global ordering: If S_1 has neighbours S_2 and S_3, with e_1 and e_2 being edges of S_1 shared with S_2 then either all of the edges S_1 shares with S_3 are on the section of perimeter that lies in an anticlockwise direction between e_1 and e_2, or they are all on the section of perimeter that lies in an anticlockwise direction between e_2 and e_1.

Although it is possible to form feasible surfaces with shared edges that violate global

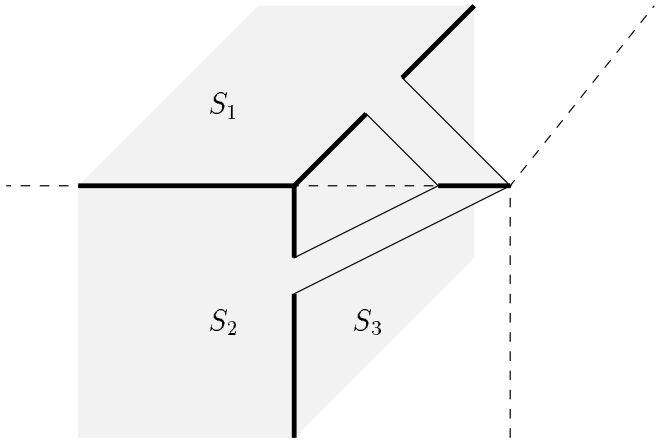


Figure 4: The result of not enforcing global ordering. The emboldened edges are shared edges and the dashed lines are the true locations of the edges being modelled.

ordering, it has been discovered experimentally that not enforcing global ordering often results in the formation of infeasible surfaces. For example, consider Figure 4 in which planar polygon segments S_1, S_2 and S_3 are all neighbours. The edges they share are emboldened and are not globally ordered. The dashed lines indicate the true locations of the edges being modelled. Thus Figure 4 is shown at high magnification. It was found that if global ordering was not enforced then structures similar to those shown in Figure 4 were sometimes formed due to small modelling errors. Thus global ordering was introduced to prevent this.

The surface model describes the surface of an object, and as such is similar to a polygon-based boundary model or polyhedral model [68] except that the surface model may not be closed. The surface model may not be closed as it only represents parts of the surface from which sensor points have been obtained. Even when the object's surface is fully modelled, the surface model may not be closed due to non-coincident corner vertices. A pair of corresponding shared edges is similar to a winged-edge structure [9] except that a pair of corresponding shared edges may not have identical vertices due to non-coincident corner vertices.

Since the surface model consists of point and planar polygon segments, it will have limited success at closely representing curved surfaces. The use of point and planar polygon segments is a compromise between surface model accuracy and the speed with which the surface model can be processed by the surface modelling and

surface following steps. If greater accuracy is required then either a greater number of point and planar polygon segments must be used or higher-order surface fitting is necessary. In both cases the computational expense of surface modelling and processing the surface model is increased.

2.3 Updating the Surface Model

The resolution with which the surface model is constructed is limited to a *linear resolution* λ^* and an *angular resolution* μ. The surface model is not expected to accurately differentiate points that are a distance less than λ from each other, nor is it expected to accurately represent angular differences that are less than μ. If the surface model contains gaps that are smaller than λ then there is no need to represent these gaps. Such gaps may be filled by applying various model segment operations:

- If the gap lies between two points on different model segments S_1 and S_2 then:
  - If either of S_1 or S_2 is a point segment then the gap may be filled by merging S_1 and S_2 with each other to form a new model segment.
  - If S_1 and S_2 are both planar polygon segments and:
    - the angle between their outwards facing normals is less than μ then the gap may be filled by merging S_1 and S_2 with each other to form a new model segment.
    - the angle between their outwards facing normals is greater than μ then the gap may be filled by forming shared edges between S_1 and S_2.
- If the gap lies between two points on the same model segment then the gap may be filled by filling *fissures* in the model segment's perimeter. Fissures will be described in Section 2.5.
- If the gap lies between points on more than two model segments then the gap may be filled by forming corner vertices.

*This is the same λ as is used in examining a model segment's eigenvalues in Section 2.2.

A set of sensor points is used to update the surface model. The sensor points are points on the surface, so they are added to the surface model by creating a model segment for each of them. The model segment data of the model segment created for a sensor point is:

$$\begin{aligned} &1, \\ &x, \quad y, \quad z, \\ &xy, \quad xz, \quad yz, \\ &x^2, \quad y^2, \quad z^2, \end{aligned}$$

where (x, y, z) are the sensor point's coordinates relative to a global coordinate frame. The eigenvalues of the covariance matrix constructed from this data are all zero, so the newly created model segment is a point segment whose description is the point (x, y, z). Once the point segments have been created, the set of sensor points may be discarded.

Associated with each sensor point is a vector that represents the sensing axis[2] of the sensor that detected the sensor point. When the sensor point was detected, the associated sensing axis must have pointed towards the inside of the surface. This information is useful when it comes to determining the outside of a planar polygon. So when a point segment is created from a sensor point, the associated sensing axis is stored with the point segment.

The addition of the newly created point segments to the surface model may form small gaps in the surface model. In an attempt to fill these gaps the following procedure is invoked:

1. Each newly created point segment is marked as having been *modified*.

2. *MERGERS*:

   (a) For each pair of distinct model segments (S_1, S_2), where S_1 has been modified:

   - If either of S_1 or S_2 is a point segment, and the distance between their descriptions is less than λ then attempt to merge S_1 and S_2 with each other.

[2] Relative to the global coordinate frame.

- If S_1 and S_2 are both planar polygon segments, the angle between their outwards facing normals is less than μ, and S_1 and S_2 are neighbours or the distance between their descriptions is less than λ then attempt to merge S_1 and S_2 with each other.

If the merger succeeds then S_1 becomes the model segment resulting from the merger, and is marked as having been modified.

(b) For each planar polygon segment S that has been modified, reposition the shared edges of S.

3. *FISSURES*:

- For each planar polygon segment S that has been modified, fill any fissures in the perimeter of S.

4. *SHARED EDGES*:

- For each pair of distinct planar polygon segments (S_1, S_2), where S_1 has been modified, if the angle between their outwards facing normals is greater than μ then compute the intersection line between the planes of S_1 and S_2. If S_1 and S_2 have non-shared edges that are a distance less than λ from the intersection line then attempt to form shared edges between S_1 and S_2.

5. *CORNER VERTICES*:

- For each planar polygon segment S that has been modified, attempt to form corner vertices between the shared edges of S.

6. For each model segment that has been modified, clear the flag that indicates that the model segment has been modified.

To reduce the computational workload, each model segment or pair of model segments is considered only once in each step of the procedure. As a result, when the procedure exits, fillable gaps may remain in the surface model. This is because new fillable gaps may be formed whenever two model segments are successfully

merged, fissures are filled, or shared edges or corner vertices are formed. To ensure that all such gaps are filled, the procedure would need to be restarted whenever two model segments are successfully merged, fissures are filled, or shared edges or corner vertices are formed. However, this would greatly increase the computational expense of the procedure.

Fillable gaps that remain after the procedure exits are usually filled during subsequent invocations of the procedure, when sensor points are obtained near such gaps. As will be described in Chapter 2, the robot is made to follow the surface using a series of short motions. Thus one set of sensor points will lie close to the next. Thus if a fillable gap is formed from a set of sensor points and remains unfilled after the procedure exits then due to the short motions of the robot, sensor points should be obtained near the gap in the next few sets of sensor points. Thus there is an opportunity for the gap to be filled by the next few invocations of the procedure. However, if the gap still remains unfilled then that part of the surface model that defines the gap will be labelled incomplete by the surface following algorithm, as will be described in Section 3.4. The surface following algorithm is designed to direct the control point to incomplete parts of the surface model. Thus there is another opportunity for sensor points to be obtained from around the unfilled gap, and so the gap may be closed during invocations of the procedure due to these sets of sensor points. If the gap still remains unfulled then the surface following algorithm will abandon it and it will probably remain unfilled.

The order in which model segments or pairs of model segments are considered may affect the resulting surface model. Similarly, the order in which the steps of the procedure are performed may also affect the resulting surface model. For example, suppose model segments S_1, S_2 and S_3 are within λ of each other then subject to suitable conditions, any of the following operations could be attempted: $merge(S_1, S_2)$, $merge(S_1, S_3)$, $merge(S_2, S_3)$, $fissure(S_1)$, $fissure(S_2)$, $fissure(S_3)$, $share(S_1, S_2)$, $share(S_1, S_3)$, $share(S_2, S_3)$ or $corners(S_1, S_2, S_3)$. The operation that yields the best results should be attempted first. However, this cannot be determined without trying all of the operations. This would be extremely time consuming. Also the best results is not always immediately apparent. Possibly

the following rules should be followed when attempting to fill gaps in the surface model:

- The gap should be between two points such that the gap is the shortest gap that can be formed using either point.
- Gaps should be filled starting with the smallest and working towards the largest.

However, applying these rules would greatly increase the computational expense of the procedure. So the model segments are simply processed in the order in which they are stored in memory. The order in which the steps are performed will now be explained.

Mergers are attempted first because they usually result in the greatest change to the surface model, that is, they usually result in the greatest change to the model segments' descriptions. Mergers produce planar polygon segments with planes that are different from those of the merged model segments. As a result, the shared edges of the planar polygon segments resulting from the mergers need to be repositioned. Repositioning of shared edges is described in Section 2.4.4. Fissures are then filled because this usually reduces the number of edges of planar polygon segments. This reduces the computational workload of the shared edges and corner vertices steps. Attempts to form shared edges are performed before attempts to form corner vertices because the greater the number of shared edges the greater the likelihood of successfully forming corner vertices.

An attempt is made to form shared edges between two planar polygon segments S_1 and S_2 when they have non-shared edges that are a distance less than λ from the intersection line. As a result, shared edges may be formed when the gap between the descriptions of S_1 and S_2 is greater than λ. However, the benefits are that the computation of the distance between the descriptions of S_1 and S_2 is avoided, and some of the computation required to form the shared edges is performed.

The merging of model segments may be seen as an on-line image segmentation algorithm (refer to Section 1.1.2). The model segments are a partitioning of the sensor points into regions to which are fitted points or planes of best fit. An attempt

is made to merge two model segments if the distance between their descriptions is small, and if they are both planar polygon segments, the angle between their outwards facing normals is small. This is similar to an adjacency constraint in image segmentation. As will be seen, the success of an attempted merger depends on how well the resulting model segment represents the sensor points of the two merged model segments. This is similar to a quality of fit constraint in image segmentation. An important difference is that the surface modelling algorithm presented in this thesis processes each sensor point only once, whereas traditional image segmentation techniques process each image point multiple times. Also most image segmentation techniques exploit the inherent structure (a grid or array) of the image points, whereas there is no such structure to sets of sensor points. This lack of structure results in much of surface modelling being devoted to the maintenance of boundaries (perimeters) of model segments.

Aside from closing gaps in the surface model, filling fissures and forming shared edges and corner vertices typically reduce the complexity of the surface model, that is, they reduce the number of edges and vertices in the surface model. As mentioned, the bulk of computation performed with the surface model involves distance computations. The complexity of this computation is proportional to the number of edges/vertices involved. Also containment and sight tests are heavily used operations. The complexity of these operations is also proportional to the number of edges/vertices involved. Thus filling fissures, and forming shared edges and corner vertices, helps to reduce the computational workload required to process the surface model. Also shared edges are of great importance to surface following for determining where to move the robot, as will be described in Chapter 3. The remainder of this chapter is devoted to describing in detail the model segment operations introduced in this section.

2.4 Merging Model Segments

When merging two distinct model segments S_1 and S_2 with each other, the resultant model segment S_3 must represent the sensor points represented by S_1 and S_2.

Firstly, the model segment data of S_1 and S_2 are merged to form the model segment data of S_3. If S_1 and S_2 have model segment data:

$$\begin{array}{ccc} m_1, & & \\ \sum_{i=1}^{m_1} x_{1i}, & \sum_{i=1}^{m_1} y_{1i}, & \sum_{i=1}^{m_1} z_{1i}, \\ \sum_{i=1}^{m_1} x_{1i}y_{1i}, & \sum_{i=1}^{m_1} x_{1i}z_{1i}, & \sum_{i=1}^{m_1} y_{1i}z_{1i}, \\ \sum_{i=1}^{m_1} x_{1i}^2, & \sum_{i=1}^{m_1} y_{1i}^2, & \sum_{i=1}^{m_1} z_{1i}^2, \end{array}$$

and

$$\begin{array}{ccc} m_2, & & \\ \sum_{i=1}^{m_2} x_{2i}, & \sum_{i=1}^{m_2} y_{2i}, & \sum_{i=1}^{m_2} z_{2i}, \\ \sum_{i=1}^{m_2} x_{2i}y_{2i}, & \sum_{i=1}^{m_2} x_{2i}z_{2i}, & \sum_{i=1}^{m_2} y_{2i}z_{2i}, \\ \sum_{i=1}^{m_2} x_{2i}^2, & \sum_{i=1}^{m_2} y_{2i}^2, & \sum_{i=1}^{m_2} z_{2i}^2, \end{array}$$

respectively, then the data are added to give the model segment data of S_3:

$$\begin{array}{ccc} m1 + m2, & & \\ \sum_{i=1}^{m_1} x_{1i} + \sum_{i=1}^{m_2} x_{2i}, & \sum_{i=1}^{m_1} y_{1i} + \sum_{i=1}^{m_2} y_{2i}, & \sum_{i=1}^{m_1} z_{1i} + \sum_{i=1}^{m_2} z_{2i}, \\ \sum_{i=1}^{m_1} x_{1i}y_{1i} + \sum_{i=1}^{m_2} x_{2i}y_{2i}, & \sum_{i=1}^{m_1} x_{1i}z_{1i} + \sum_{i=1}^{m_2} x_{2i}z_{2i}, & \sum_{i=1}^{m_1} y_{1i}z_{1i} + \sum_{i=1}^{m_2} y_{2i}z_{2i}, \\ \sum_{i=1}^{m_1} x_{1i}^2 + \sum_{i=1}^{m_2} x_{2i}^2, & \sum_{i=1}^{m_1} y_{1i}^2 + \sum_{i=1}^{m_2} y_{2i}^2, & \sum_{i=1}^{m_1} z_{1i}^2 + \sum_{i=1}^{m_2} z_{2i}^2. \end{array}$$

From this set of model segment data a covariance matrix $\mathbf{C}_3$ is constructed, and the eigenvalues and eigenvectors of $\mathbf{C}_3$ are computed.

Depending on the types of S_1 and S_2, different algorithms are used to obtain the description of S_3. Also additional conditions must be satisfied before the merger can succeed. If the merger is successful then S_1 and S_2 are deleted from the surface model and replaced by S_3, otherwise S_1 and S_2 remain unchanged. The algorithms for merging different types of model segments are described in the remainder of this section.

2.4.1 Merging Two Point Segments

When S_1 and S_2 are both point segments, their sensing axes must not oppose one another. The sensing axes indicate the direction in which the inside of the surface lies. If the sensing axes indicate that the inside of the surface lies in two opposing directions then S_1 and S_2 should not be merged. So if the angle between the sensing axes of S_1 and S_2 is greater than $90°$ then the merger fails.

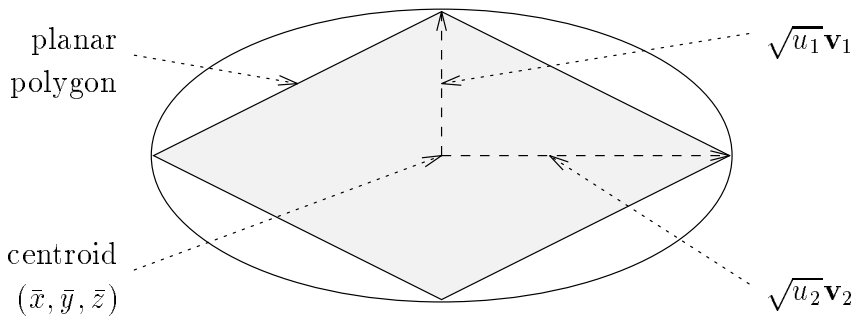


Figure 5: The planar polygon resulting from the merger of two point segments.

The result of merging two point segments may be a point segment or a planar polygon segment. To determine the type of S_3, the eigenvalues of $\mathbf{C}_3$, and the variances s^2_{pt} and s^2_{pp} are examined, as described in Section 2.2.

If S_3 is a point segment then its description is the centroid of the sensor points it must represent. If the sensor points' variance about the centroid is too great then S_1 and S_2 should not be merged. So if $s^2_{\text{pt}} \geq \lambda^2$ then the merger fails. As S_3 is to be a point segment it requires a sensing axis, so the sensing axes of S_1 and S_2 are added together, and the resulting vector becomes the sensing axis of S_3.

If S_3 is a planar polygon segment then its description is a planar polygon lying on the plane of best fit to the sensor points S_3 must represent. If the sensor points' variance about the plane is too great then S_1 and S_2 should not be merged. So if $s^2_{\text{pp}} \geq \lambda^2$ then the merger fails.

A planar set of points can be approximated by the ellipse whose centre is the points' centroid and whose radii are given by the two largest eigenvalues and their associated eigenvectors of the points' covariance matrix. Thus the initial planar polygonal representation of a set of sensor points could be such an ellipse. A diamond-shaped planar polygon is constructed that approximates the ellipse, as shown in Figure 5. The planar polygon's vertices are positioned at the following points:

$$(\bar{x}, \bar{y}, \bar{z}) + \sqrt{u_1}\,\mathbf{v}_1,$$
$$(\bar{x}, \bar{y}, \bar{z}) + \sqrt{u_2}\,\mathbf{v}_2,$$
$$(\bar{x}, \bar{y}, \bar{z}) - \sqrt{u_1}\,\mathbf{v}_1,$$
$$(\bar{x}, \bar{y}, \bar{z}) - \sqrt{u_2}\,\mathbf{v}_2,$$

where u_1 and u_2 are the two largest eigenvalues of $\mathbf{C}_3$, and $\mathbf{v}_1$ and $\mathbf{v}_2$ are their

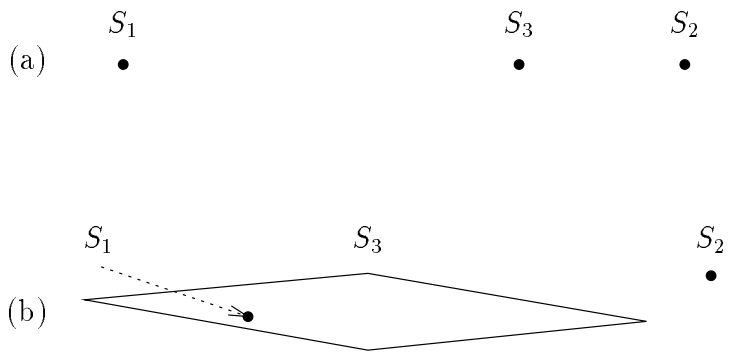


Figure 6: Examples of the merger of two point segments. (a) The result is a point segment as $s^2_{\text{pt}} < s^2_{\text{pp}}$. (b) The result is a planar polygon segment as $s^2_{\text{pt}} \geq s^2_{\text{pp}}$.

associated eigenvectors, respectively. The vertices are connected by edges to form a perimeter whose direction is anticlockwise relative to the outwards facing normal of S_3. To determine the outwards facing normal, the sensing axes of S_1 and S_2 are used. The sensing axes indicate the direction in which the inside of the surface lies, so they are added together and the opposite of the resulting vector is projected onto the normal of the plane of S_3. This gives the outwards facing normal of S_3.

An extra copy of the plane of S_3 is stored by storing the centroid and outwards facing normal of S_3. This extra copy, called the *initial plane*, remains fixed and is used in subsequent mergers involving S_3.

Figure 6 shows examples of the merger of two point segments. In Figure 6(a) the sensor points represented by S_1 and S_2 are such that $s^2_{\text{pt}} < s^2_{\text{pp}}$, and so the resulting model segment S_3 is a point segment. Note that S_3 is closer to S_2 because S_2 represents more sensor points than S_1. In Figure 6(b) the sensor points represented by S_1 and S_2 are such that $s^2_{\text{pt}} \geq s^2_{\text{pp}}$, and so the resulting model segment S_3 is a planar polygon segment. Note that the centroid of S_3 is closer to S_1 because S_1 represents more sensor points than S_2.

2.4.2 Merging a Point Segment and a Planar Polygon Segment

Let S_1 and S_2 be the point segment and planar polygon segment, respectively. The sensing axis of S_1, and the outwards facing normal of S_2 must not be in the same direction. The sensing axis and outwards facing normal indicate the directions in which the inside and outside, respectively, of the surface lies. If the sensing axis and outwards facing normal indicate that the inside and outside of the surface lie in the same direction then S_1 and S_2 should not be merged. So if the angle between the sensing axis of S_1, and the outwards facing normal of S_2 is less than $90°$ then the merger fails.

The result of merging a point segment and a planar polygon segment may only be a planar polygon segment. So if $s^2_{\text{pt}} < s^2_{\text{pp}}$ then the merger fails. The description of S_3 is a planar polygon lying on the plane of best fit to the sensor points S_3 must represent. If the sensor points' variance about the plane is too great then S_1 and S_2 should not be merged. So if $s^2_{\text{pp}} \geq \lambda^2$ then the merger fails. The outwards facing normal of S_3 is obtained by projecting the outwards facing normal of S_2 onto the normal of the plane of S_3.

Consider a series of mergers:

$$\begin{aligned} T_1 + U_1 &\rightarrow U_2, \\ T_2 + U_2 &\rightarrow U_3, \\ T_3 + U_3 &\rightarrow U_4, \\ T_4 + U_4 &\rightarrow U_5, \end{aligned}$$

where the notation $T_i + U_i \rightarrow U_{i+1}$ means point segment T_i and planar polygon segment U_i are merged to form planar polygon segment U_{i+1}. The plane of U_1 may be significantly different from that of U_5, as shown in Figure 7. To prevent this from occurring the initial plane of S_2 is compared with the plane of S_3. If they are significantly different then the merger fails, otherwise the initial plane of S_3 is copied from the initial plane of S_2. The initial plane of S_2 is significantly different from the plane of S_3 if either of the following are satisfied:

- The angle between their outwards facing normals is greater than μ.

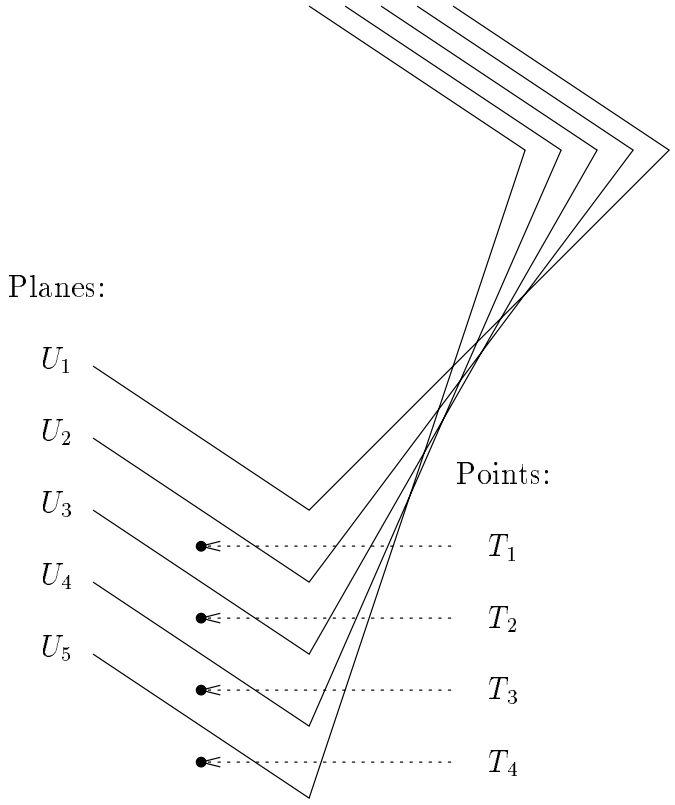


Figure 7: A series of mergers may result in a significantly different plane of best fit.

- The centroid of S_3 is a perpendicular distance greater than λ from the initial plane of S_2.

The planar polygon description of S_3 is obtained by merging the descriptions of S_1 and S_2. The planar polygon lies on the plane of S_3, so the point and planar polygon descriptions of S_1 and S_2, respectively, are orthogonally projected onto this plane. The projected point and projected planar polygon are then merged to form the planar polygon of S_3. If the projected planar polygon contains the projected point then the planar polygon of S_3 is simply the projected planar polygon. If not then the planar polygon of S_3 is formed by connecting the projected planar polygon's perimeter to the projected point. To connect the perimeter to the projected point, a section of the perimeter is extended to the projected point, as shown in Figure 8.

To determine the perimeter section to extend to the projected point, a pair of distinct perimeter points p_1 and p_2 is chosen. The perimeter section P that lies

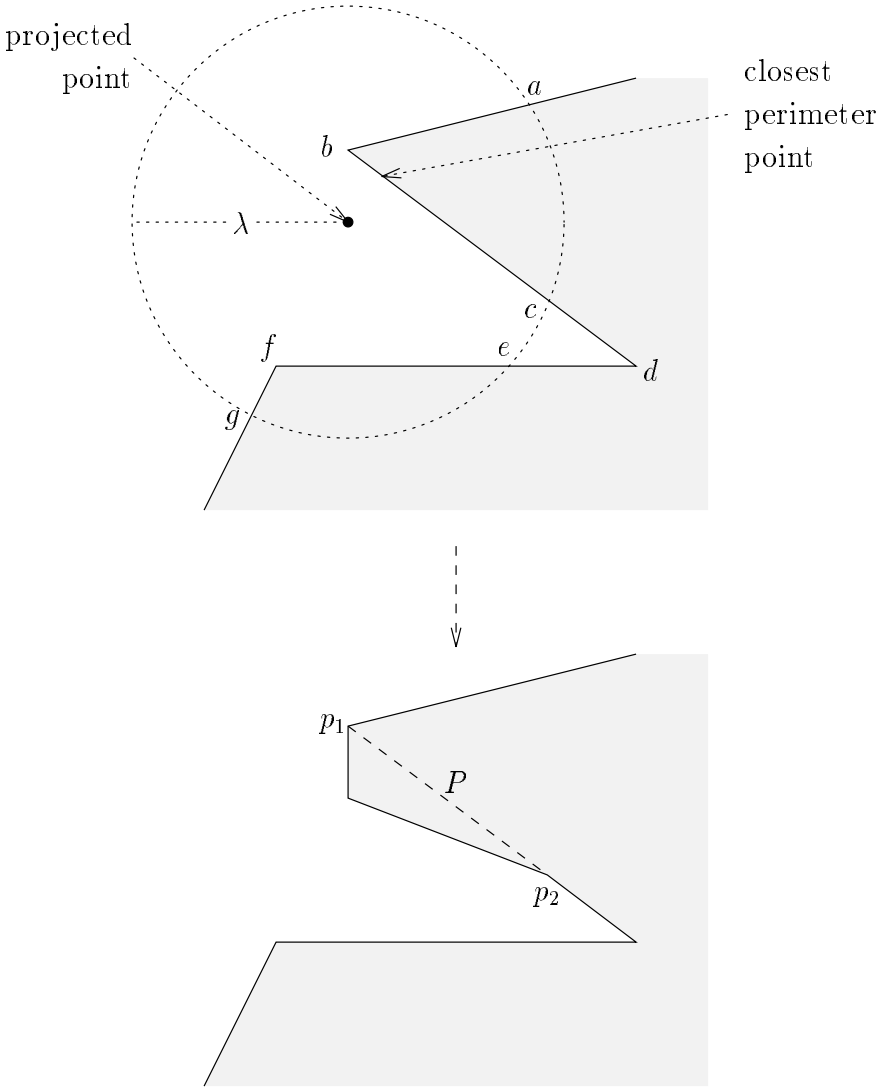


Figure 8: Merging a projected point with a projected perimeter.

in an anticlockwise direction between p_1 and p_2 is then extended to the projected point. Perimeter points p_1 and p_2 are chosen to satisfy the following:

- Perimeter points p_1, p_2 and vertices on P must be a distance less than λ from the projected point. This ensures that P is entirely within a distance λ of the projected point.

- The perimeter point that is closest to the projected point must lie on P. This ensures that P is as close to the projected point as possible.

- Perimeter points p_1 and p_2 must be able to *sight* the projected point relative

to the projected perimeter. A point can sight another if the straight line that lies between them does not cross the perimeter. This ensures that when P is extended to the projected point, the new edges formed will not cross the resulting perimeter, and P will be contained by the resulting planar polygon. In Figure 8, perimeter points that lie on the perimeter section *bdf* can sight the projected point.

- P must be as long as possible.
- No shared edges nor portions of them may lie on P. This ensures that when P is extended to the projected point, no shared edge is affected.

To reduce the computational workload involved in choosing p_1 and p_2, only vertices and the perimeter points that lie at the ends of perimeter sections that are a distance less than λ from the projected point, are considered as possible candidates for p_1 and p_2. This reduces the workload at the expense of the possibility of missing a valid pair of perimeter points. In Figure 8, only perimeter points $a, \ldots, g$ would be considered as possible candidates for p_1 and p_2, and b and c are valid choices for p_1 and p_2, respectively.

Once p_1 and p_2 have been chosen, P is extended to the projected point as follows:

1. Vertices are formed at p_1, p_2 and the projected point. If p_1 or p_2 coincide with a vertex then that vertex is used rather than forming a new one.
2. Edges are formed that connect the vertices at p_1 and p_2 with the projected point's vertex.
3. P is removed.

If a valid pair of perimeter points is not found then the merger fails. Therefore if the closest perimeter point lies on a shared edge, as in Figure 9(b), then the merger would fail. However, in such cases the merger is made to succeed with the resulting perimeter of S_3 being the projected perimeter. Alternatively, the shared edge could be unshared and the projected perimeter extended to the projected point. Edge unsharing simply removes the shared status of an edge, as will be

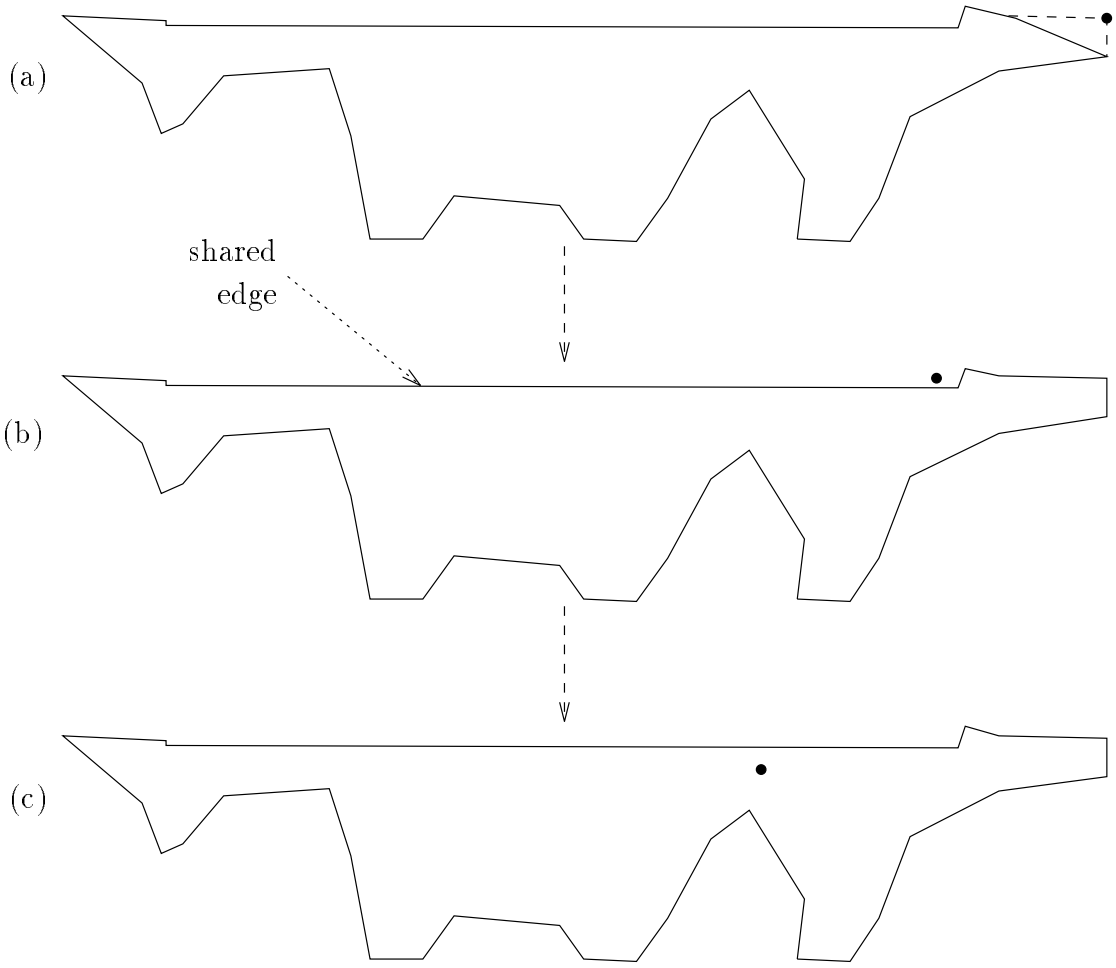


Figure 9: An example of a series of mergers of projected points with projected perimeters. (a) The projected perimeter is extended to the projected point. The dashed lines indicate the new edges that are formed. (b) The closest perimeter point lies on a shared edge. (c) The projected point is contained by the projected polygon.

described in Section 2.8. However, when this approach was taken it was found that the unshared edge was usually subsequently reformed, returning the perimeter to its original shape. Thus this process was circumvented by simply using the projected perimeter as the result of the merger. So if the closest perimeter point lies on a shared edge then the perimeter of S_3 is the projected perimeter.

Figure 9 shows an example of a series of mergers of projected points with projected perimeters. In Figure 9(a) the projected perimeter is extended to the projected point. In Figure 9(b) the closest perimeter point lies on a shared edge, so the perimeter of S_3 is the projected perimeter. In Figure 9(c) the projected point is

contained by the projected planar polygon, so the perimeter of S_3 is the projected perimeter.

2.4.3 Merging Two Planar Polygon Segments

The result of merging two planar polygon segments S_1 and S_2 may only be a planar polygon segment. So if $s^2_{\text{pt}} < s^2_{\text{pp}}$ then the merger fails. The description of S_3 is a planar polygon lying on the plane of best fit to the sensor points S_3 represent. If the sensor points' variance about the plane is too great then S_1 and S_2 should not be merged. So if $s^2_{\text{pp}} \geq \lambda^2$ then the merger fails. The outwards facing normal of S_3 is obtained by projecting the outwards facing normal of S_1 onto the normal of the plane of S_3. The initial plane of S_3 is then copied from the plane of S_3.

The planar polygon description of S_3 is obtained by merging the descriptions of S_1 and S_2. The planar polygon lies on the plane of S_3, so the planar polygons of S_1 and S_2 are orthogonally projected onto this plane. The projected planar polygons P_1 and P_2 obtained from S_1 and S_2, respectively, are then merged to form the planar polygon of S_3. There are two cases to consider: P_1 and P_2 do not overlap, or P_1 and P_2 overlap. These two cases are discussed in the following subsections.

Merging Non-Overlapping Planar Polygons

If P_1 and P_2 do not overlap then the planar polygon of S_3 is formed by connecting the perimeters of P_1 and P_2. To connect the perimeters, a perimeter section of P_1 is extended to a perimeter section of P_2, or *vice versa*, as shown in Figure 10. To determine the perimeter sections, a pair of distinct perimeter points p_1 and p_2 is chosen from one of the perimeters, and a pair of distinct perimeter points q_1 and q_2 is chosen from the other perimeter. The perimeter section P that lies in an anticlockwise direction between p_1 and p_2, is extended to the perimeter section Q that lies in an anticlockwise direction between q_1 and q_2. Perimeter points p_1, p_2, q_1 and q_2 are chosen to satisfy the following:

- Perimeter points p_1 and q_2 must be a distance less than λ from each other, and perimeter points p_2 and q_1 must be a distance less than λ from each other.

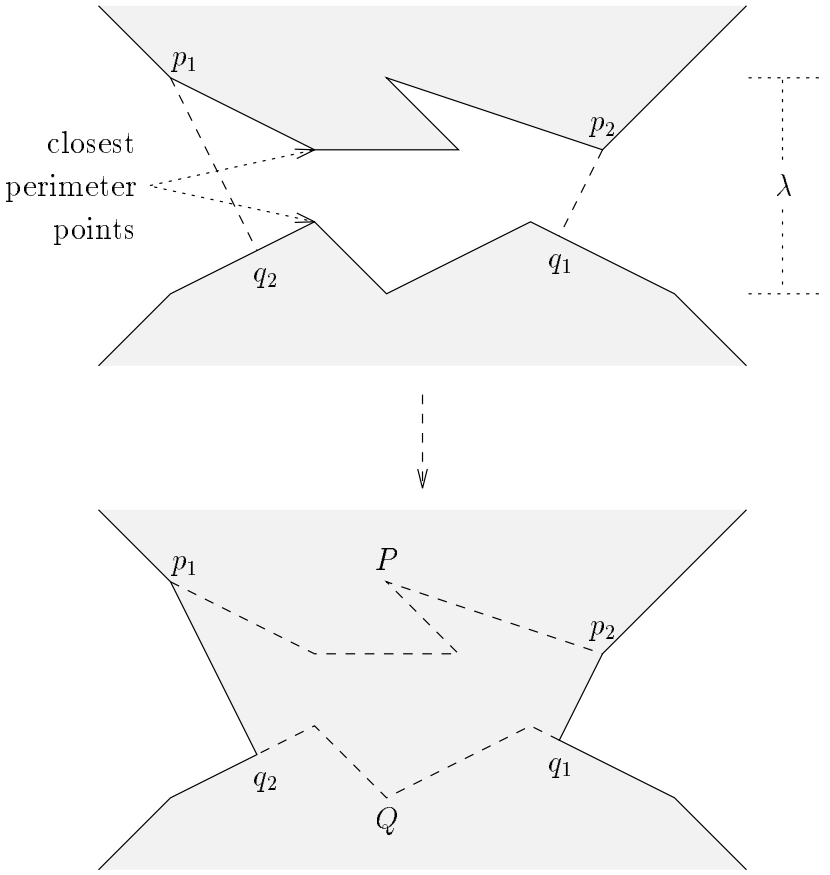


Figure 10: Merging two non-overlapping planar polygons.

Also edges on P must be a distance less than λ from Q, and edges on Q must be a distance less than λ from P. This ensures that all perimeter points on P are within a distance λ of Q, and *vice versa*.

- The perimeter point on P_1 that is closest to P_2 must lie on P, and the perimeter point on P_2 that is closest to P_1 must lie on Q. This ensures that P and Q are as close to each other as possible.

- Perimeter point p_1 must be able to sight q_2 relative to the perimeters of P_1 and P_2, and perimeter point p_2 must be able to sight q_1 relative to the perimeters of P_1 and P_2. This ensures that when P is extended to Q, the new edges formed will not cross the resulting perimeter, and P and Q will be contained by the resulting planar polygon.

- P must be as long as possible.

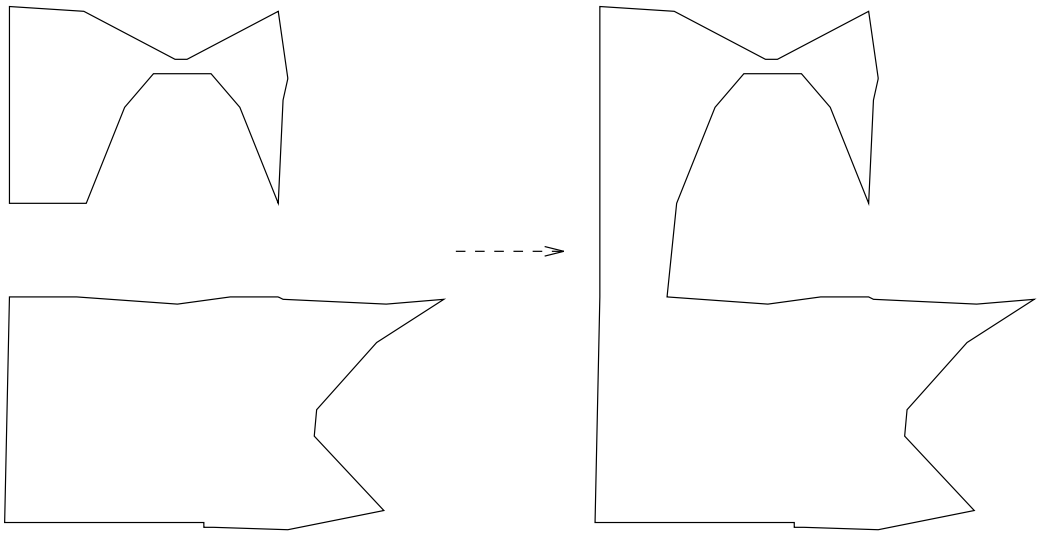


Figure 11: An example of a merger of non-overlapping planar polygons.

To reduce the computational workload involved in choosing the perimeter points, only vertices are considered as possible candidates for p_1 and p_2, and on the other perimeter only the points closest to p_1 and p_2 are considered as possible candidates q_2 and q_1, respectively, as shown in Figure 10. This reduces the workload at the expense of the possibility of missing valid perimeter points.

Once the perimeter points have been chosen, P is extended to Q as follows:

1. Vertices are formed at q_1 and q_2. If q_1 or q_2 coincide with a vertex then that vertex is used rather than forming a new one.

2. Edges are formed that connect the vertex at p_1 with the vertex at q_2, and the vertex at p_2 with the vertex at q_1.

3. P and Q are removed. If a shared edge or portion of one lies on P or Q then the edge is unshared before the perimeter section is discarded. Section 2.8 describes the unsharing of shared edges.

If a valid pair of vertices p_1 and p_2 is not found on either perimeter then the merger fails. Figure 11 shows an example of a merger of non-overlapping planar polygons.

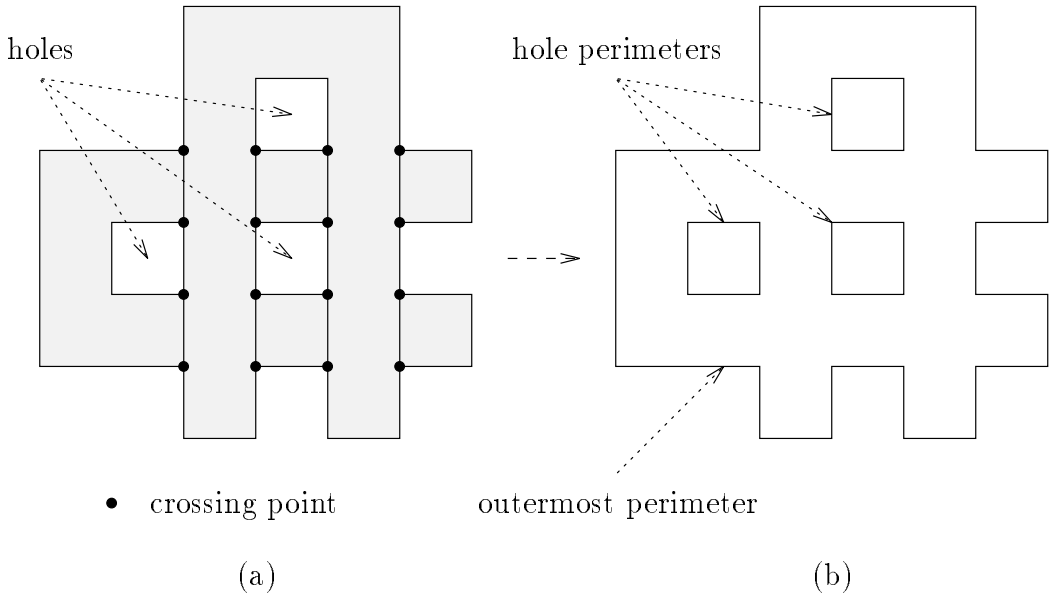


Figure 12: Overlapping perimeters. (a) The crossing points and holes. (b) The outermost perimeter and hole perimeters.

Merging Overlapping Planar Polygons

If P_1 and P_2 overlap then parts of each planar polygon lie inside the other. If one planar polygon lies wholly inside the other then the planar polygon of S_3 is the projected planar polygon that contains the other. If not then the perimeters of P_1 and P_2 must cross each other. A *crossing point* is defined as a point at which the perimeters cross each other, as shown in Figure 12(a). If the perimeters coincide for some distance before crossing each other then the coincident section represents a single crossing point that can be placed at any point along the coincident section.

Successive crossing points that lie along a perimeter, delimit perimeter sections that lie either inside or outside of the other planar polygon. The planar polygon of S_3 is formed by connecting perimeter sections of P_1 that lie outside of P_2, with perimeter sections of P_2 that lie outside of P_1. This is achieved by forming vertices at the crossing points. If P_1 and P_2 overlap to form *holes*, as shown in Figure 12(a), then multiple perimeters are formed. If multiple perimeters are formed then the outermost perimeter is the perimeter not contained by any of the other perimeters, and a hole's perimeter is contained by the outermost perimeter, as shown in Figure 12(b). The single perimeter description of a planar polygon cannot represent

planar polygons that contain holes, so the holes are discarded if possible. A hole can be discarded if its perimeter contains fissures that when filled cause the hole's perimeter to collapse. Section 2.5 describes the filling of fissures. If a hole's perimeter does not collapse then the merger fails. If all holes' perimeters collapse then the perimeter of S_3 is the outermost perimeter.

In forming the perimeter of S_3, perimeter sections are discarded, either because they lie inside the other planar polygon or because they border holes. If a shared edge or portion of one lies on a perimeter section to be discarded then the edge is unshared before the perimeter section is discarded. Section 2.8 describes the unsharing of shared edges.

Figure 13 shows examples of mergers of overlapping planar polygons. In Figure 13(a) no holes are formed by the overlapping planar polygons, whereas in Figure 13(b) the planar polygons overlap to form holes that are collapsed during the merger.

Merging Neighbours

Shared edges of S_1 and S_2 may be retained on the planar polygon of S_3. Thus neighbours of S_1 and S_2 may become neighbours of S_3. If S_1 and S_2 are neighbours of each other then before their merger is attempted their neighbour relationship is dissolved by unsharing all of the edges they share, that is, they are no longer neighbours as they no longer share any edges. Thus if the merger succeeds then S_3 will not share edges with itself, and if the merger fails then S_1 and S_2 are no longer neighbours. It is desirable that S_1 and S_2 no longer be neighbours since the angle between their outwards facing normals is less than μ.

The ordering of any shared edges retained by S_3 is checked to ensure that both the local ordering and global ordering constraints are satisfied. If local ordering is not satisfied, that is, edges that S_3 shares with one of its neighbours are disordered, then the neighbour relationship with that neighbour is dissolved. If global ordering is not satisfied, that is, sets of edges that S_3 shares with a pair of its neighbours are disordered, then the neighbour relationships with those two neighbours are dissolved.

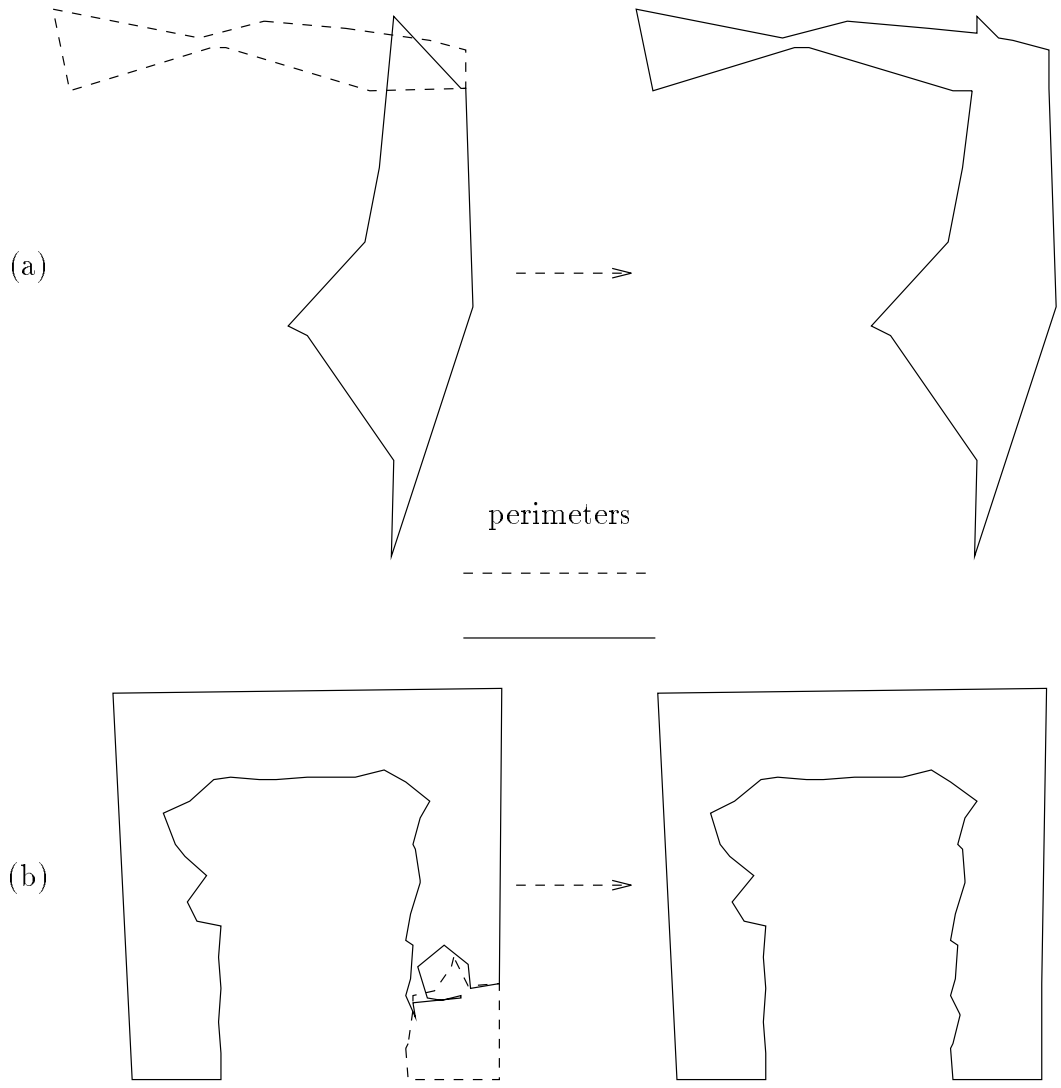


Figure 13: Examples of mergers of overlapping planar polygons. (a) No holes are formed by the overlap. (b) A hole is formed by the overlap.

2.4.4 Repositioning Shared Edges

If S_1 and S_2 are successfully merged to form a planar polygon segment S_3 then S_3 may have inherited neighbours of S_1 and S_2. Suppose S_3 has inherited a neighbour N of S_1. Then the plane of S_3 is unlikely to coincide with the plane of S_1, therefore the intersection line $\mathbf{l}$ between the planes of S_3 and N is unlikely to coincide with the intersection line $\mathbf{l}_1$ between the planes of S_1 and N. The edges that S_3 shares with N should lie along $\mathbf{l}$. However, they have been inherited by orthogonal projection onto the plane of S_3. In general, this will not result in the shared edges lying

along $\mathbf{l}$. The edges that N shares with S_3 should also lie along $\mathbf{l}$. However, they currently lie along $\mathbf{l}_1$. So the shared edges of S_3, and corresponding shared edges of the neighbours of S_3 must be repositioned to lie along the appropriate intersection lines. The shared edges are repositioned by repositioning their vertices. These vertices may be corner vertices or non-corner vertices.

Firstly, the intersection lines between the planes of S_3 and each of its neighbours are computed. If the angle between the outwards facing normals of S_3 and one of its neighbours N is 180°, that is, the planes of S_3 and N are parallel and their outwards facing normals are in opposite directions, then the intersection line between the planes of S_3 and N cannot be formed, and so the neighbour relationship between S_3 and N is dissolved. This can occur, for example, when a pair of neighbours intersect at an angle close to 180°, and the plane of one of the model segments is changed due to a merger, such that the angle between the model segments becomes 180°. In such cases the neighbour relationship is dissolved. Note that the angle between the outwards facing normals of S_3 and each of its neighbours will always be greater than μ. This is because an attempt will have been made to merge neighbours with outwards facing normals that form an angle less than μ, as described in Section 2.3. When an attempt is made to merge two neighbouring model segments, their neighbour relationship is dissolved, as described in Section 2.4.3.

Once the intersection lines have been computed, the shared edges' corner vertices are repositioned, as shown in Figure 14. This simply involves intersecting the appropriate intersection lines and repositioning the corner vertex at the intersection point. If the intersection lines are parallel then the intersection point does not exist, so the corner vertex and its corresponding corner vertices are converted to non-corner vertices. The conversion of a corner vertex to a non-corner vertex is described in Section 2.8.

Once the corner vertices have been repositioned, the shared edges' non-corner vertices are orthogonally projected onto the appropriate intersection lines. The shared edges then lie along the appropriate intersection lines. The section of overlap between each pair of corresponding shared edges is determined. If a pair of

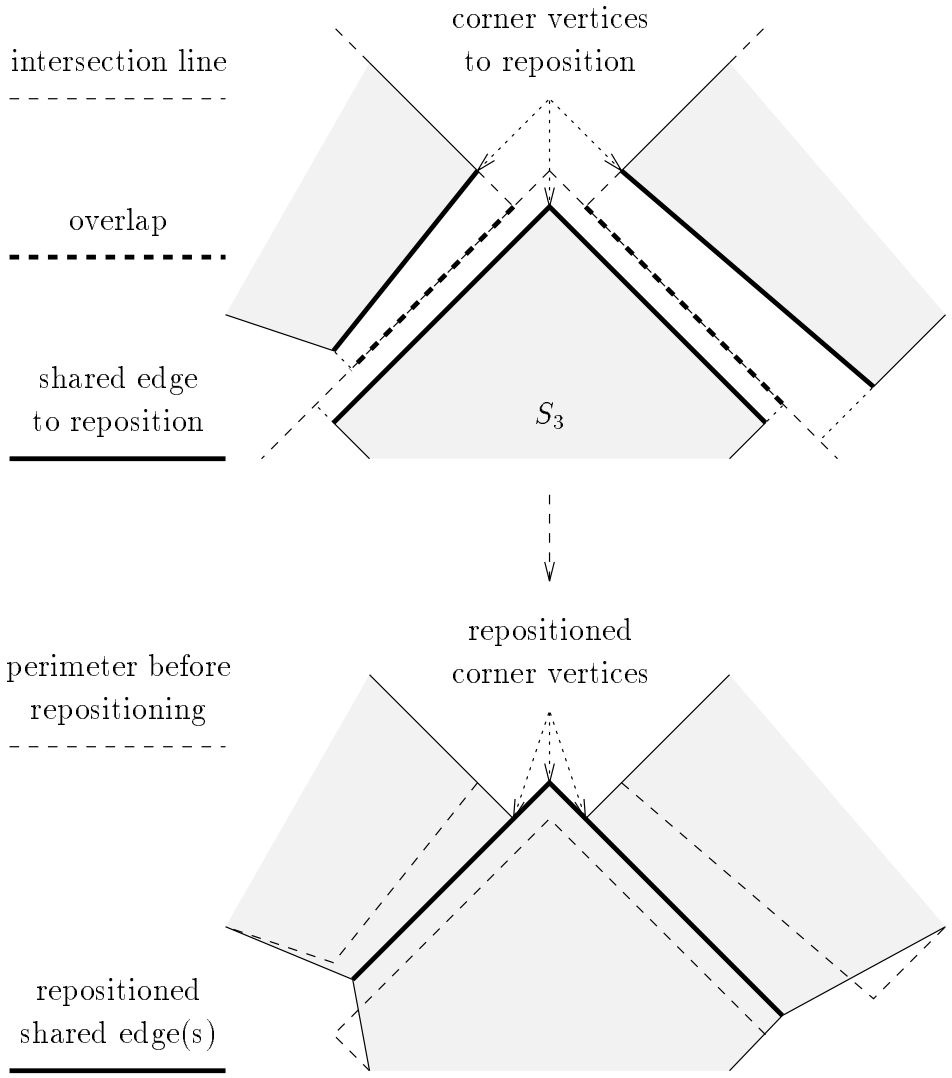


Figure 14: Repositioning corner vertices and shared edges.

corresponding shared edges do not overlap then they are unshared. If a repositioned non-corner vertex lies outside of the section of overlap then it is moved to an end of the section of overlap such that it coincides with the appropriate vertex of the corresponding shared edge, as shown in Figure 14.

Repositioning the shared edges may result in perimeters whose edges cross each other. To prevent this, each repositioned shared edge is checked as follows: If v_1, v_2, v_3, and v_4 are consecutive vertices along a perimeter with v_2 and v_3 being the vertices of a repositioned shared edge then the following pairs of vertices must be able to sight each other relative to the perimeter: v_1 and v_2, v_2 and v_3, and v_3 and

v_4. If any of these pairs of vertices cannot sight each other then the repositioned shared edge is unshared and v_2 and v_3 are returned to their original positions.

As mentioned in Section 2.2, the anticlockwise direction along a shared edge depends upon the concavity between the model segments that share the edge, and the model segments' outwards facing normals. If the anticlockwise direction along a repositioned shared edge is incorrect relative to this information then the shared edge is unshared and returned to its original position.

2.5 Filling Fissures

Three consecutive vertices v_1, v_2 and v_3 of a perimeter, where neither of the edges v_1v_2 and v_2v_3 are shared edges, may form the following types of fissures:

I. The external angle between the edges is less than 180°, and the distance between v_1 and v_3 is less than λ, as shown in Figure 15(a). The external angle is measured on the outside of the perimeter.

II. The external angle between the edges is less than 90°, and the distance between v_1 and the edge v_2v_3 is less than λ, as shown in Figure 15(b).

III. The external angle between the edges is less than 90°, and the distance between v_3 and the edge v_1v_2 is less than λ, as shown in Figure 15(c).

Four consecutive vertices v_1, v_2, v_3 and v_4, where none of the edges v_1v_2, v_2v_3 and v_3v_4 are shared edges, and the length of the edge v_2v_3 is less than λ, may form the following types of fissures:

IV. The external angle between edges v_1v_2 and v_3v_4 is less than 180°, and the distance between v_1 and v_4 is less than λ, as shown in Figure 15(d).

V. A type II fissure formed with vertices v_1, v_3 and v_4, as shown in Figure 15(e).

VI. A type III fissure formed with vertices v_1, v_2 and v_4, as shown in Figure 15(f).

The different types of fissure are filled as follows (refer to Figure 15):

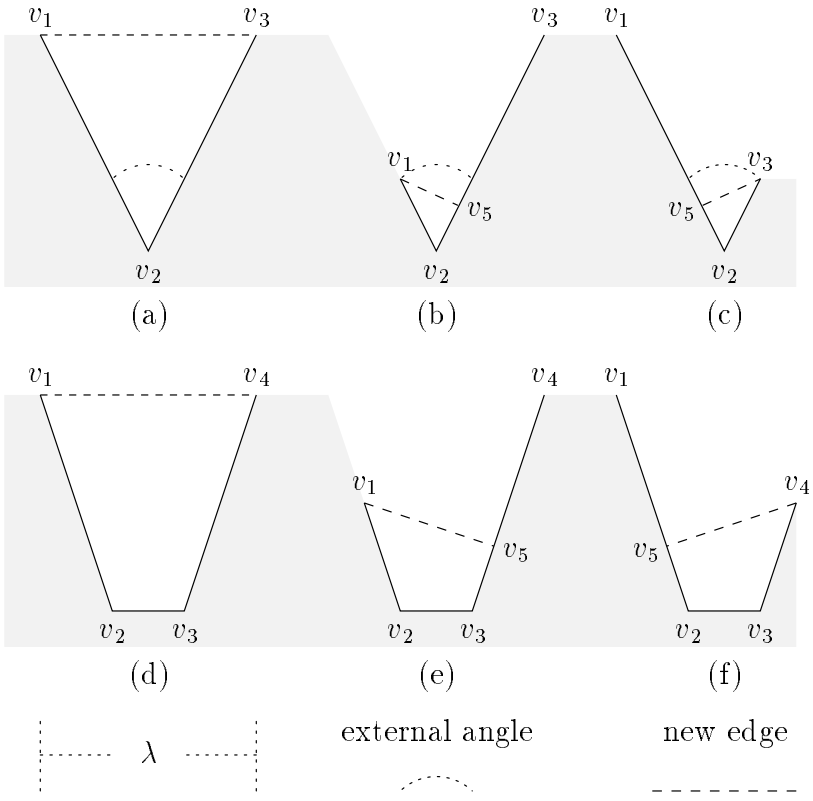


Figure 15: Types of fissure: (a) Type I. (b) Type II. (c) Type III. (d) Type IV. (e) Type V. (f) Type VI.

I. An edge is formed between v_1 and v_3, and the perimeter section $v_1v_2v_3$ is removed.

II. A vertex v_5 is formed on the edge v_2v_3 at the point closest to v_1. Then an edge is formed between v_1 and v_5, and the perimeter section $v_1v_2v_5$ is removed.

III. A vertex v_5 is formed on the edge v_1v_2 at the point closest to v_3. Then an edge is formed between v_5 and v_3, and the perimeter section $v_5v_2v_3$ is removed.

IV. An edge is formed between v_1 and v_4, and the perimeter section $v_1v_2v_3v_4$ is removed.

V. A vertex v_5 is formed on the edge v_3v_4 at the point closest to v_1. Then an edge is formed between v_1 and v_5, and the perimeter section $v_1v_2v_3v_5$ is removed.

VI. A vertex v_5 is formed on the edge v_1v_2 at the point closest to v_4. Then an edge is formed between v_5 and v_4, and the perimeter section $v_5v_2v_3v_4$ is removed.

To fill fissures in a perimeter, each set of consecutive vertices v_1, v_2, v_3 and v_4 of the perimeter is considered. A set of vertices may form a number of different types of fissure. The order in which fissure types are checked is IV, V, VI, I, II, III. This ordering is used because filling a type IV fissures removes more perimeter than filling a type V or VI fissure, filling a type V or VI fissure removes more perimeter than filling a type I fissure, and filling a type I fissures removes more perimeter than filling a type II or III fissure. Originally, only fissures of type I, II and III were considered. However, it was found that type IV, V and VI fissures often remained unfilled as a result. Thus fissure filling was extended to include type IV, V and VI fissures.

When an edge is to be formed to fill a fissure, the vertices at either end of the edge must be able to sight one another. If not then the edge is not formed and the fissure is not filled. This ensures that when a fissure is filled, the resulting perimeter does not cross itself. Filling a fissure may form another fissure. So when a new edge is formed to fill a fissure, it is checked to see whether it forms another fissure. If so then that fissure is filled, and so on. Figure 16 shows an example of a series of fissures being filled.

When filling fissures in order to collapse a hole's perimeter, as described in Section 2.4.3, the restrictions that edges of the fissure may not be shared, and that the vertices of the new edge must be able to sight each other, are removed. When filling fissures in a hole's perimeter, the objective is to remove the perimeter, not form a new perimeter, and so these restrictions are not important.

The filling of fissures could also be achieved using the morphological closing operation [45]. The closing operation does not affect convex sections of the perimeter of a region but does fill narrow concavities (fissures). A structuring element whose diameter is no greater that λ would need to be used in the closing operation.

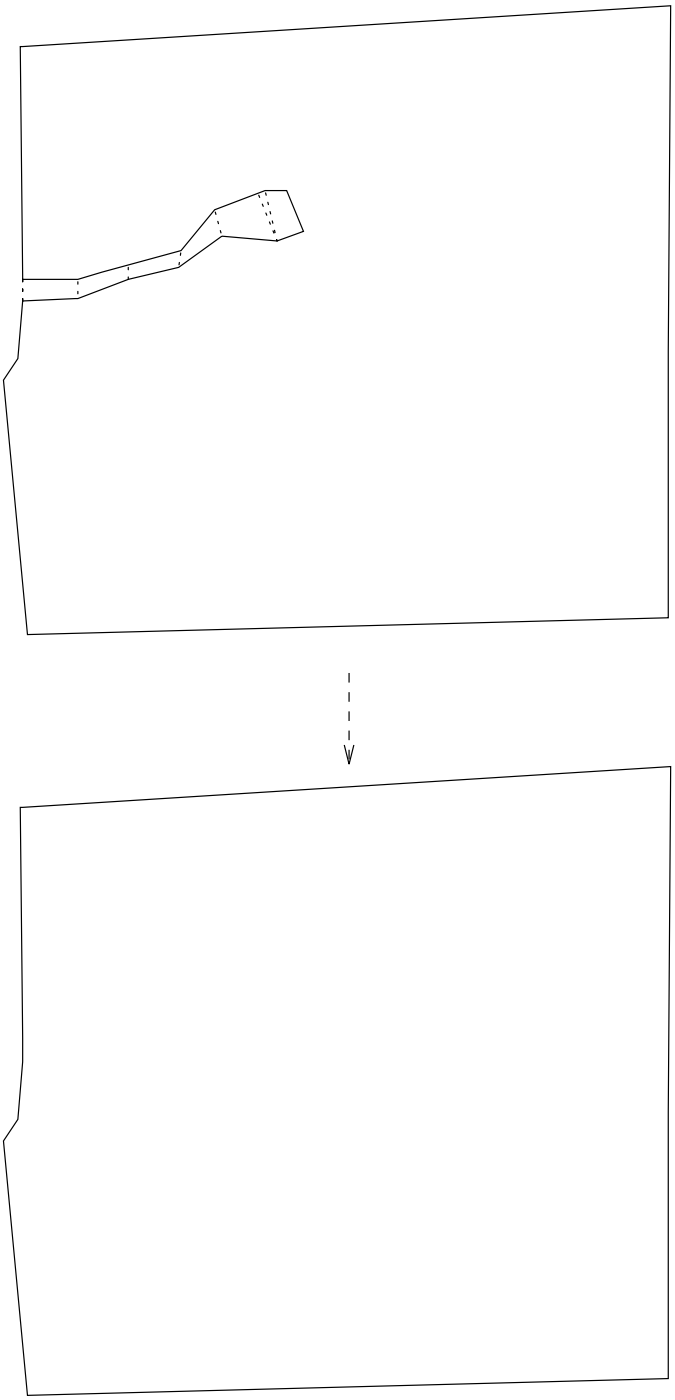


Figure 16: Filling a series of fissures. The dotted lines are the edges formed to fill fissures.

2.6 Forming Shared Edges

To form edges that are shared between two distinct planar polygon segments S_1 and S_2, sections of the perimeters of S_1 and S_2 are extended to the intersection line

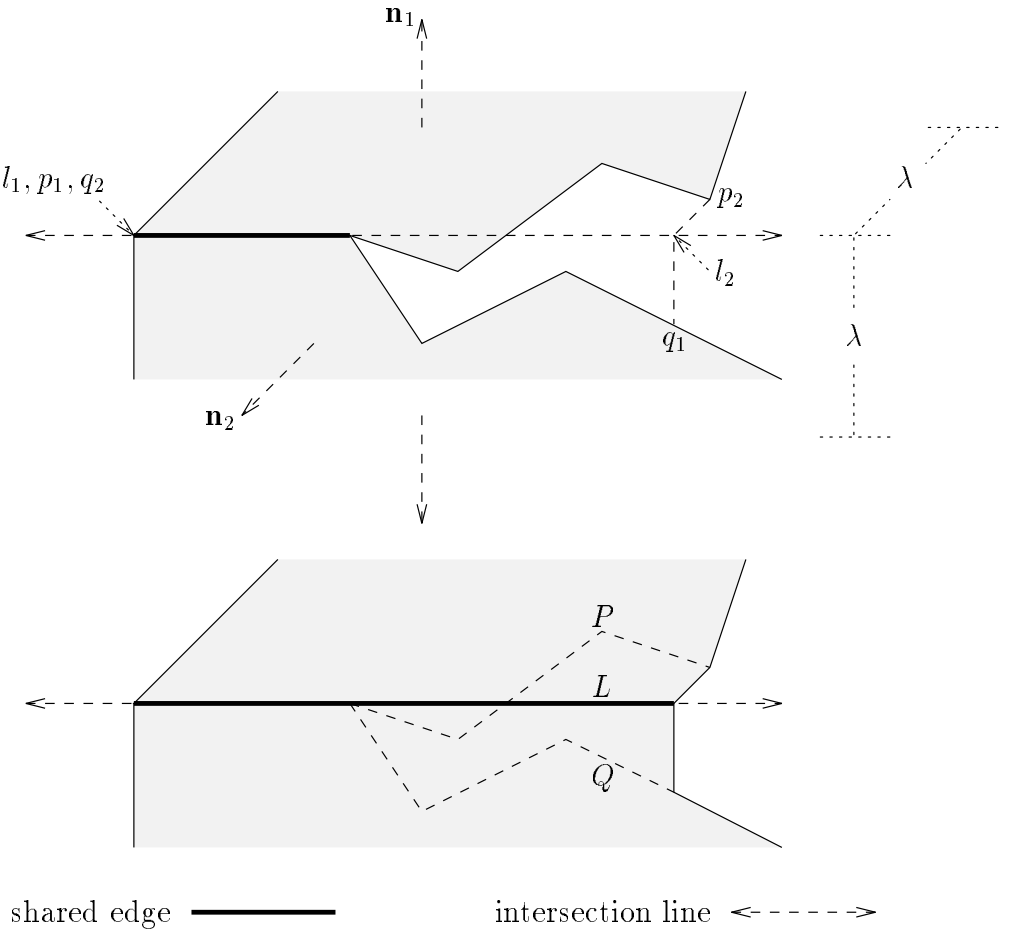


Figure 17: Forming a pair of corresponding shared edges.

between the planes of S_1 and S_2. Let P_1 and P_2 be the planar polygons of S_1 and S_2, respectively. To form a pair of corresponding shared edges, a perimeter section of P_1, and a perimeter section of P_2 are extended to a section L of the intersection line, as shown in Figure 17. The intersection line may pass through P_1 or P_2, so when a perimeter section is extended to the intersection line, portions of P_1 or P_2 may be removed, as shown in Figure 17.

To determine the perimeter sections, a pair of distinct perimeter points p_1 and p_2 is chosen from P_1, and a pair of distinct perimeter points q_1 and q_2 is chosen from P_2. Perimeter points p_1 and q_2 orthogonally project to the same point l_1 on the intersection line, and perimeter points p_2 and q_1 orthogonally project to the same point l_2 on the intersection line. L is the section of intersection line that lies between l_1 and l_2. Alternatively, p_1 and q_2 may coincide with corresponding corner vertices. In this case p_1 and q_2 lie on the intersection line but do not necessarily

coincide with each other. In such cases l_1 is made to coincide with p_1 or q_2 such that L is longer. Similarly, p_2 and q_1 may coincide with corresponding corner vertices. In such cases l_2 is made to coincide with p_2 or q_1 such that L is longer. To form a pair of corresponding shared edges the perimeter section P that lies in an anticlockwise direction between p_1 and p_2 is extended to L, and the perimeter section Q that lies in an anticlockwise direction between q_2 and q_1 is also extended to L. Perimeter points p_1, p_2, q_1 and q_2 are chosen to satisfy the following:

- Perimeter points p_1, p_2, q_1, q_2, vertices on P, and vertices on Q must be a distance less than λ from L. This ensures that P and Q are entirely within a distance λ of L.

- When vertices on P and Q are orthogonally projected to the intersection line, their projections must lie on L. This ensures that P and Q entirely project to L.

- Points l_1 and l_2 must be distinct and if the concavity between S_1 and S_2 is convex then the vector from l_1 to l_2 must be parallel to $\mathbf{n}_1 \times \mathbf{n}_2$, where $\mathbf{n}_1$ and $\mathbf{n}_2$ are the outwards facing normals of S_1 and S_2, respectively. If the concavity is concave then the vector must be parallel to $\mathbf{n}_2 \times \mathbf{n}_1$. This ensures that when the shared edges are formed, the anticlockwise directions along them are correct relative to $\mathbf{n}_1$, $\mathbf{n}_2$ and the concavity. Section 2.6.1 presents methods for determining the concavity between S_1 and S_2.

- Perimeter points p_1 and p_2 must be able to sight l_1 and l_2, respectively, relative to the perimeter of P_1, and q_1 and q_2 must be able to sight l_2 and l_1, respectively, relative to the perimeter of P_2. This ensures that when P and Q are extended to L, the new edges formed will not cross the resulting perimeters.

- Point l_1 must be able to sight l_2, relative to the perimeter sections formed by removing P from P_1, and Q from P_2. This ensures that when the shared edges are formed, they will not cross the resulting perimeters. Figure 18 illustrates what can happen if l_1 cannot sight l_2.

- P and Q must be as long as possible.

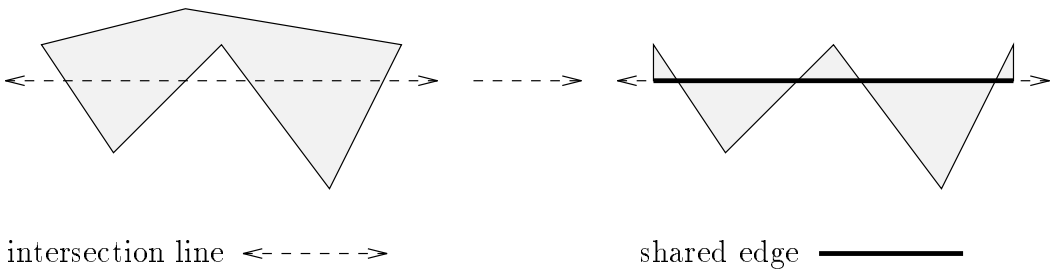


Figure 18: The shared edge crosses the perimeter.

- The only shared edges lying on P and Q are shared between S_1 and S_2. This ensures that when P and Q are extended to L, edges that S_1 and S_2 share with other neighbours are unaffected.

- If a shared edge lies on P then its corresponding shared edge must lie on Q, and *vice versa*.

- There is at least part of a non-shared edge along P and Q. This ensures that new shared edges are formed.

To reduce the computational workload involved in choosing the perimeter points, only vertices are considered as possible candidates for the perimeter points that determine l_1 and l_2, that is, at least one of p_1 and q_2 is a vertex, and at least one of p_2 and q_1 is a vertex. This reduces the workload at the expense of the possibility of missing valid perimeter points.

Once the perimeter points have been chosen, P is extended to L as follows:

1. Vertices are formed at p_1 and p_2. If p_1 or p_2 coincide with a vertex then that vertex is used rather than forming a new one.

2. Vertices are formed at l_1 and l_2, and a shared edge is formed between these vertices.

3. Edges are formed that connect the vertex at p_1 with the vertex at l_1, and the vertex at p_2 with the vertex at l_2.

4. If p_1 coincides with a vertex of an edge that is shared between S_1 and S_2, as shown in Figure 17, then that vertex lies on L. In such a case a vertex is not

formed at l_1, and an edge is not formed between p_1 and l_1. Instead the vertex at p_1 becomes a vertex of the new shared edge. A similar case occurs if p_2 coincides with a vertex of an edge that is shared between S_1 and S_2.

5. P is removed.

Similarly, Q is extended to L to form the corresponding shared edge, except that edges are formed that connect the vertex at q_1 with the vertex at l_2, and the vertex at q_2 with the vertex at l_1. The attempt to form shared edges between S_1 and S_2 is complete once valid perimeter points cannot be found. If no shared edges were formed between S_1 and S_2 then S_1 or S_2 may be deleted, as will be described in Section 2.9.

The ordering of the newly formed shared edges is checked to ensure that both the local ordering and global ordering constraints are satisfied for S_1 and S_2. If local ordering is not satisfied then the neighbour relationship between S_1 and S_2 is dissolved. If global ordering is not satisfied then the neighbour relationships not satisfying global ordering are dissolved. One of these relationships will be between S_1 and S_2, and the other will be between S_1 or S_2 and another neighbour. If local ordering or global ordering is not satisfied then no further attempts to form shared edges between S_1 and S_2 are made until the next set of sensor points is received.

Figure 19 shows an example of the formation of a pair of shared edges. The concavity between the model segments is convex; the outside angle between their planes being about 270°. Note that the perimeter sections extended to the intersection line contain two pairs of corresponding shared edges. These shared edges are replaced by the new pair that is formed.

2.6.1 Determining Concavity

If S_1 and S_2 are neighbours then the concavity between them is given by the concavity of the outside angle between their planes at any edge they share, as described in Section 2.2. Note that there may only be one type of concavity between S_1 and S_2. If S_1 and S_2 are not neighbours then the concavity between them could be set according to the following rules:

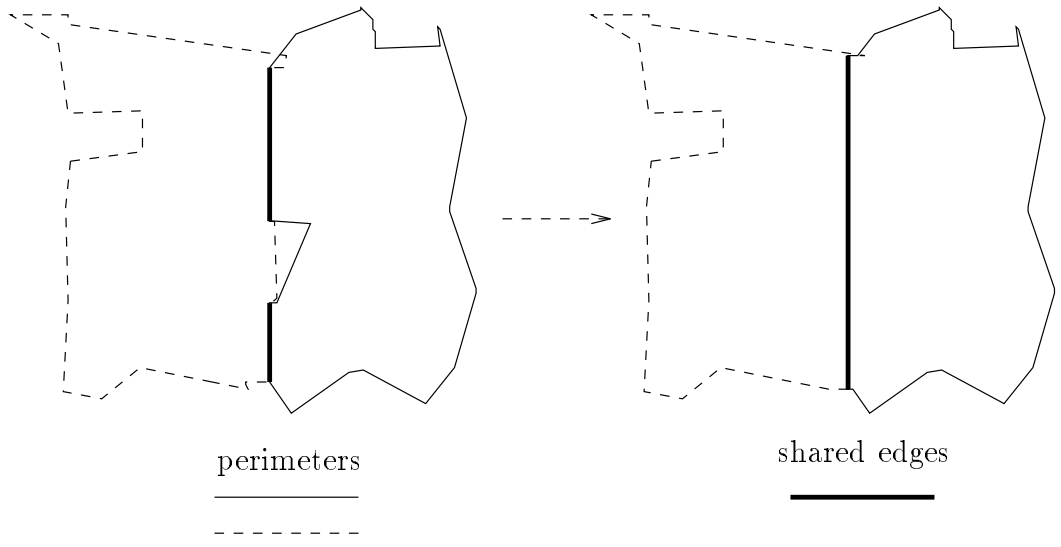


Figure 19: An example of the formation of a pair of corresponding shared edges.

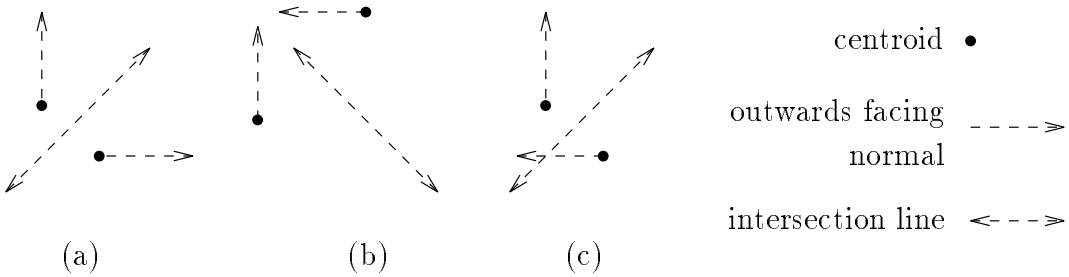


Figure 20: Determining concavity using centroids. (a) Convex. (b) Concave. (c) Undecided.

- The concavity is convex if each model segment's centroid is on the inside of the plane of the other model segment, as shown in Figure 20(a).
- The concavity is concave if each model segment's centroid is on the outside of the plane of the other model segment, as shown in Figure 20(b).

However, these rules do not consider the following cases (refer to Figure 20(c)):

- The centroid of S_1 is on the outside of the plane of S_2, and the centroid of S_2 is on the inside of the plane of S_1.
- The centroid of S_2 is on the outside of the plane of S_1, and the centroid of S_1 is on the inside of the plane of S_2.

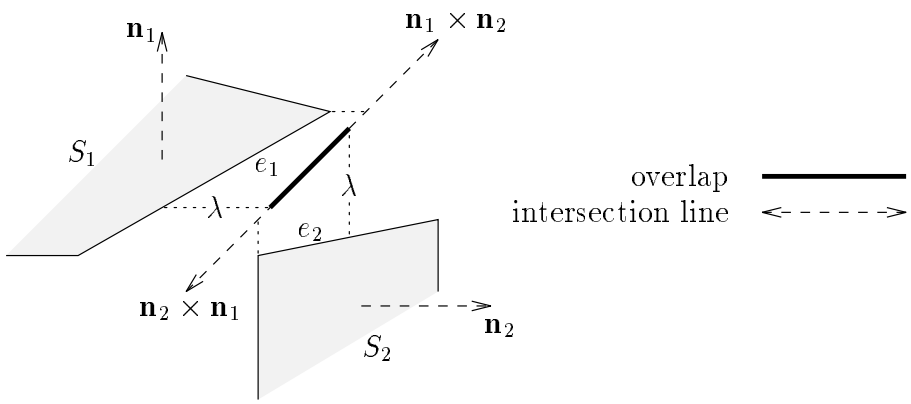


Figure 21: Determining concavity.

Clearly, if S_1 and S_2 are not neighbours then to determine the concavity, more than the planes and centroids must be considered.

Recall from Section 2.6 that depending on the concavity between S_1 and S_2, the vector from l_1 to l_2 must be parallel to either $\mathbf{n}_1 \times \mathbf{n}_2$ or $\mathbf{n}_2 \times \mathbf{n}_1$. Thus the concavity affects the formation of shared edges. If S_1 and S_2 are not neighbours then the concavity could be set such that the greatest total length of shared edges is formed. However, to do this would be extremely computationally expensive as the conditions given in Section 2.6 would have to be tested twice; once for each type of concavity. Instead a simpler test is performed on the edges of S_1 and S_2. Each pair of edges (e_1, e_2) is considered, where e_1 and e_2 are edges of S_1 and S_2, respectively. For the pair of edges to have an influence on the concavity decision, e_1 and e_2 must satisfy the following (refer to Figure 21):

- Part of e_1 and part of e_2 must be a distance less than λ from the intersection line. This is because the perimeter sections extended to the intersection line must be entirely within a distance λ of the intersection line.

- The parts of e_1 and e_2 that are a distance less than λ from the intersection line are orthogonally projected to the intersection line. These projections must overlap. This is because the perimeter sections extended to the intersection line must project to the same section of the intersection line. The length of the overlap is determined.

- The anticlockwise direction along the projection of e_1 must be opposite to the anticlockwise direction along the projection of e_2. This is because the anticlockwise directions along a pair of corresponding shared edges must be opposite to each other.

If e_1 and e_2 satisfy these conditions then a shared edge could possibly be formed from perimeter sections containing (parts of) e_1 and e_2 if:

- the concavity is convex and the anticlockwise direction along the projection of e_1 is parallel to $\mathbf{n}_1 \times \mathbf{n}_2$, or
- the concavity is concave and the anticlockwise direction along the projection of e_1 is parallel to $\mathbf{n}_2 \times \mathbf{n}_1$.

So if e_1 and e_2 satisfy the aforementioned conditions and the projection of e_1 is parallel to $\mathbf{n}_1 \times \mathbf{n}_2$ then the length of the overlap is added to a register kept for the convex case, and if the projection of e_1 is parallel to $\mathbf{n}_2 \times \mathbf{n}_1$ then the length of the overlap is added to a register kept for the concave case. Once all pairs of edges (e_1, e_2) have been considered, the concavity between S_1 and S_2 is set to the concavity whose register has the greatest value. It is possible for the registers to have equal (non-zero) values. In such cases the concavity remains undecided, and no attempt to form shared edges between S_1 and S_2 is made until the next set of sensor points is received. If the concavity is undecided then S_1 or S_2 may be deleted, as will be described in Section 2.9.

The algorithm for determining concavity is not foolproof; in carefully contrived cases it can assign the wrong concavity. For example, Figure 22 shows two model segments that appear to meet convexly. However, the algorithm assigns them a concave intersection because local to their overlap on the intersection line they meet concavely. Thus only concave shared edges would be allowed to be formed between the two model segments. By considering local features, this algorithm is the opposite of the technique, described at the beginning of this section, that considers global features such as the model segments' centroids when determining concavity. In practice it was found that cases such as Figure 22 seldom occurred and so the more local algorithm gave better results.

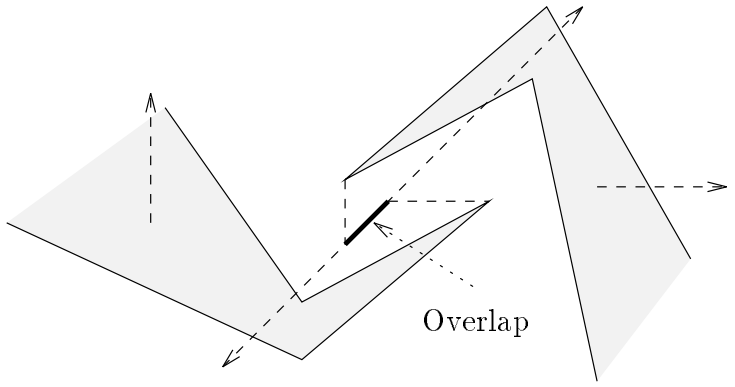


Figure 22: Two model segments that meet convexly but are assigned concavity.

This method of determining the concavity is still fairly computationally expensive. This is why the concavity is given by the outside angle between the planes of S_1 and S_2 if S_1 and S_2 are neighbours, and why neighbours may only have one type of concavity.

2.7 Forming Corner Vertices

Suppose that a planar polygon segment S has a pair of distinct shared edges e_1 and e_2, such that:

- The next shared edge following e_1 in an anticlockwise direction along the perimeter of S is e_2.
- Edges e_1 and e_2 are shared with different neighbours of S, such that e_1 lies along the intersection line $\mathbf{l}_1$ between the planes of S and one of its neighbours, and e_2 lies along the intersection line $\mathbf{l}_2$ between the planes of S and another of its neighbours.

Let v_1 be the vertex at the end of e_1 in an anticlockwise direction, v_2 be the vertex at the start of e_2 in an anticlockwise direction, and P be the perimeter section that lies in an anticlockwise direction between v_1 and v_2. To form a corner vertex between e_1 and e_2, P is extended to the point at which $\mathbf{l}_1$ and $\mathbf{l}_2$ intersect, as shown in Figure 23. The point at which $\mathbf{l}_1$ and $\mathbf{l}_2$ intersect is called the corner point. Edges e_1 and e_2 must satisfy the following:

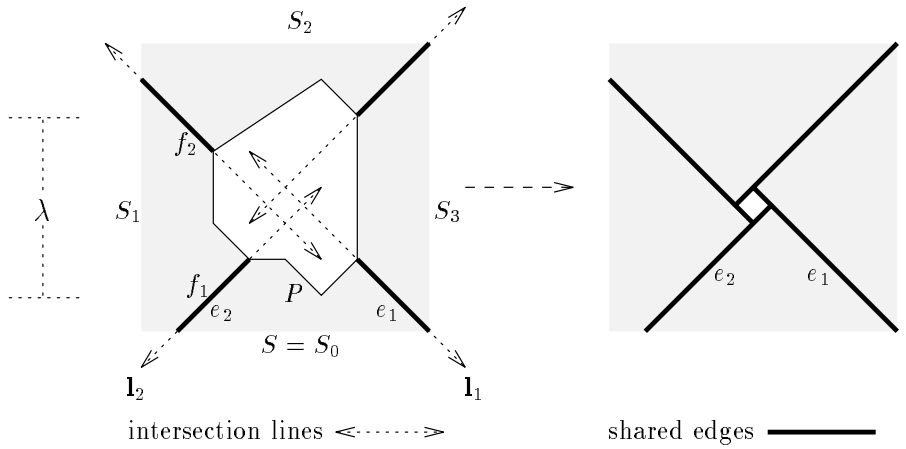


Figure 23: Forming corner vertices.

- There must not already be a corner vertex between e_1 and e_2.
- Each point on P must be a distance less than λ from either $\mathbf{l}_1$ or $\mathbf{l}_2$.
- Intersection lines $\mathbf{l}_1$ and $\mathbf{l}_2$ must not be parallel, that is, the corner point must exist.
- There must be a point on P that is a distance less than λ from the corner point.
- When v_1 and v_2 are moved to the corner point, the anticlockwise directions along e_1 and e_2 must be preserved. This ensures that when the corner vertex is formed, the anticlockwise directions along e_1 and e_2 are correct relative to the concavities between S and its neighbours, and the outwards facing normals of S and its neighbours (refer to Section 2.2).
- The intersection point must be able to sight v_1 and v_2 relative to the perimeter section formed by removing P from the perimeter. This ensures that when the corner vertex is formed, edges e_1 and e_2 will not cross the resulting perimeter.

If a corner vertex is to be formed between e_1 and e_2 then corresponding corner vertices must also be formed on the neighbours of S, their neighbours, and so on. So similar conditions must also be satisfied for these model segments. That is, if S_1

is the neighbour with which S shares e_2, and S_1 has a pair of distinct shared edges f_1 and f_2, with:

- f_1 being the shared edge corresponding to e_2,
- f_2 being the next shared edge following f_1 in an anticlockwise direction along the perimeter of S_1, and
- f_1 and f_2 being shared with different neighbours of S_1,

then the aforementioned conditions must be satisfied for f_1 and f_2. Similarly, the conditions must also be satisfied for the neighbour S_2 with which S_1 shares f_2, and so on to S_n with which S shares e_1. So the conditions must be satisfied for a set of distinct model segments $\{(S =)S_0, S_1, S_2, \ldots, S_n\}$, where S_i and S_{i+1} are neighbours, and S_0 and S_n are neighbours. If the conditions cannot be satisfied for such a set of neighbouring model segments then the attempt to form corner vertices fails. In Figure 23 the conditions are satisfied for the set of neighbouring model segments $\{S_0, S_1, S_2, S_3\}$.

If the conditions are satisfied then corner vertices are formed for each of the model segments $S_0, \ldots, S_n$. A corner vertex is formed between shared edges e_1 and e_2 as follows:

1. Vertex v_1 of edge e_1 is moved to the corner point, where it replaces v_2 as the start vertex of e_2, that is, e_1 and e_2 are joined at v_1.
2. P, including vertex v_2, is removed.

The intersection lines between the neighbouring model segments may not all intersect at the same point, so the corner vertices may not be coincident, as shown in Figure 23. Figure 24 shows an example of the formation of a set of corner vertices.

2.8 Unsharing Edges

Suppose that S_1 and S_2 are neighbours, and e_1 and e_2 are shared edges of S_1 and S_2, respectively, such that e_1 and e_2 form a pair of corresponding shared edges. If

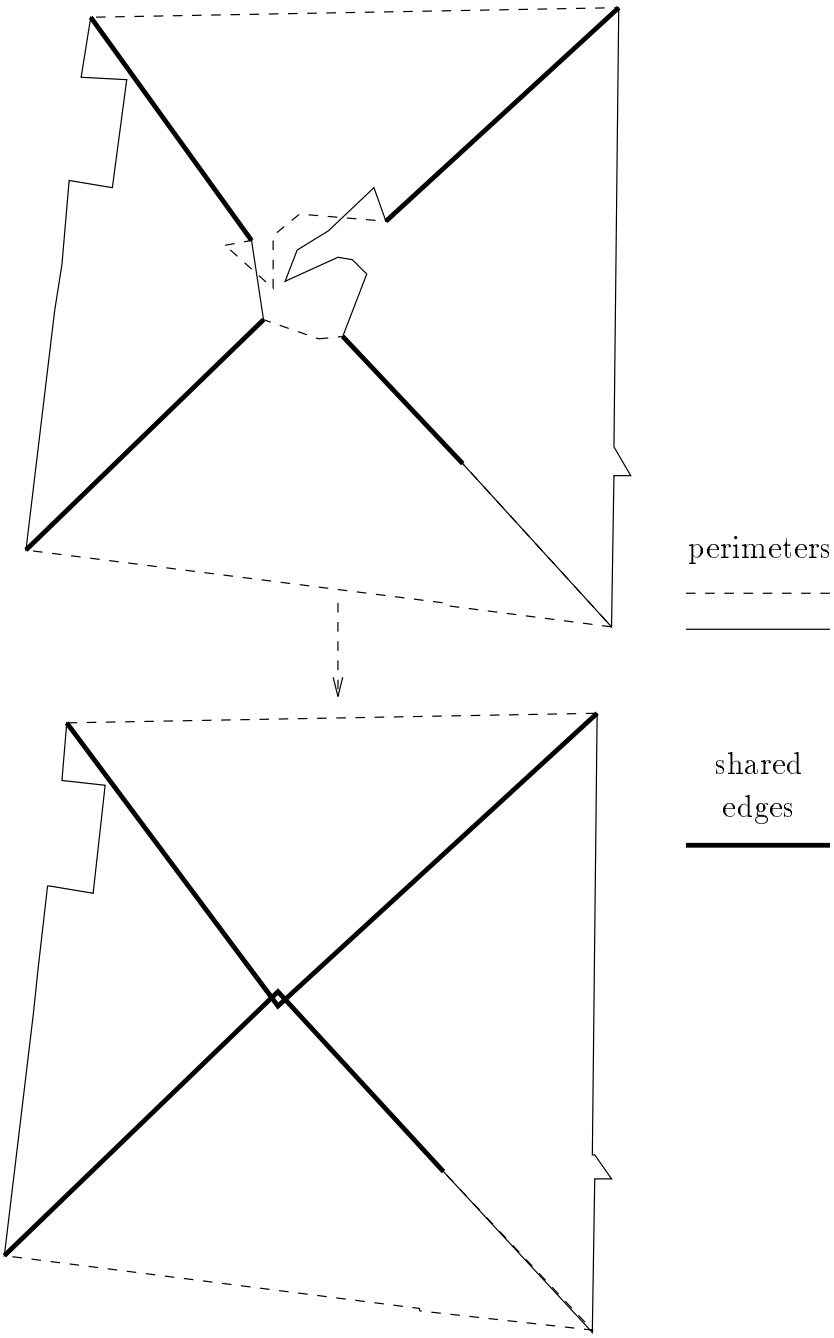


Figure 24: An example of the formation of a set of corner vertices.

e_1 is to be *unshared* then e_2 must also be unshared, and *vice versa*. A shared edge is unshared simply by converting it to a non-shared edge. If either of the vertices of the edge is a corner vertex then the corner vertex becomes a non-corner vertex. If a corner vertex becomes a non-corner vertex then its corresponding corner vertices also become non-corner vertices.

A corner vertex becoming a non-corner vertex may result in the non-corner vertex joining two shared edges. If these shared edges are subsequently inherited as the result of a merger then they will be repositioned, as described in Section 2.4.4. So the non-corner vertex that joins the two shared edges will be repositioned twice; once for each of the shared edges that it joins. Thus one of the shared edges will not be correctly repositioned. To avoid this problem, if a corner vertex becomes a non-corner vertex that joins two shared edges then a duplicate of the non-corner vertex is created, and an edge formed between the non-corner vertex and its duplicate. The non-corner vertex becomes an end of one of the shared edges, and the duplicate vertex becomes an end of the other shared edge.

If e_1 and e_2 are the only edges shared between S_1 and S_2, and they are unshared then S_1 and S_2 are no longer neighbours, that is, the neighbour relationship between S_1 and S_2 is dissolved. Conversely, if a neighbour relationship is to be dissolved then all edges shared between the neighbours are unshared.

2.9 Deleting Model Segments

Occasionally, a small planar polygon segment is formed between two or more other model segments, such that it cannot be merged with any of them, as shown in Figure 25. These small planar polygon segments are *redundant* in the sense that their removal from the surface model improves the surface model's description of the surface. Redundant model segments usually represent only a few sensor points, and if an attempt is made to form shared edges between a redundant model segment and some other model segment then usually the concavity is undecided or no shared edges are formed. Thus if an attempt is made to form shared edges between two model segments S_1 and S_2, where S_1 and S_2 are not neighbours, and either the concavity is undecided or no shared edges are formed then S_1 or S_2 could be redundant. Assume without loss of generality that S_1 represents fewer sensor points than S_2, that is, $n_1 < n_2$, where n_1 and n_2 are the number of sensor points represented by S_1 and S_2, respectively. S_1 is redundant if $n_1 < \psi$ and $n_1/n_2 < \phi$, where ψ and ϕ are preset threshold values.

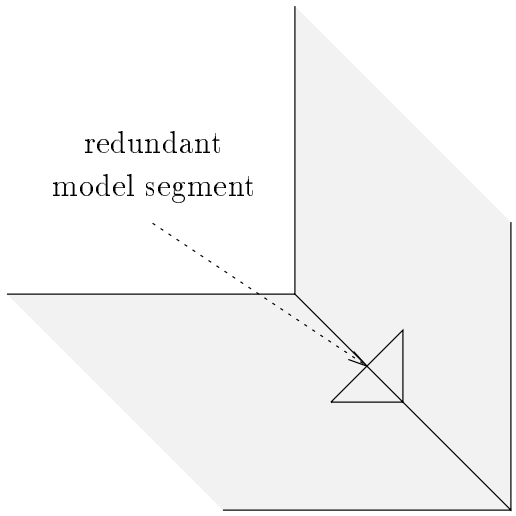


Figure 25: A redundant model segment.

If a model segment is determined to be redundant then it is deleted from the surface model. If the redundant model segment has any neighbours then those neighbour relations are dissolved before the redundant model segment is deleted. The sensor points represented by the deleted model segment are no longer represented in the surface model. If ψ is too large then when a redundant model segment is deleted a large amount of sensor data is lost. If ψ is too small then many model segments will be deleted too early before they have had a chance to develop into redundant or non-redundant model segments. If ϕ is too large then the value of n_2 has little impact on the redundancy decision. If ϕ is too small then n_2 must be very large for the redundant model segment to be deleted.

Suitable threshold values were determined empirically by surface modelling with redundancy checking but without deleting redundant model segments. For each redundant model segment in the final surface model the first redundancy check performed for that model segment was determined and the corresponding values of n_1 and n_2 obtained. The averages of these n_1's and n_2's for the redundant model segments of several final surface models were used to determine the values of ψ and ϕ. These values are given in Section 4.3.

The test for redundancy is simple, and as a result non-redundant model segments are sometimes incorrectly classified as redundant, and are subsequently deleted.

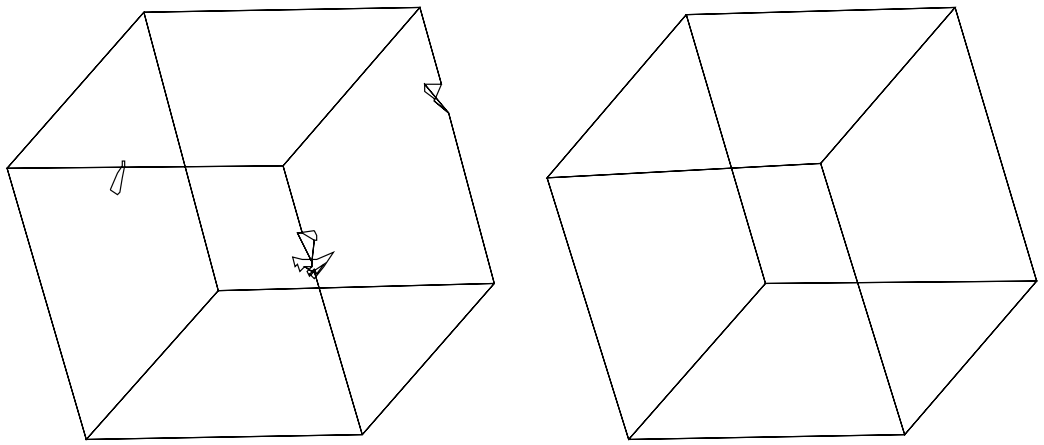


Figure 26: (a) Redundant model segments have not been deleted. (b) Redundant model segments have been deleted.

Also some redundant model segment fail to be detected, and so are not deleted. However, the test detects most redundant model segments, and is computationally inexpensive.

Figure 26 show two examples of completed surface models of a cube. In Figure 26(a) redundant model segments have not been deleted, and in Figure 26(b) they have been deleted. The surface model without redundant model segments is obviously better than the surface model with redundant model segments.

2.10 Summary

The surface model is a set of model segments. Each model segment has either a point or planar polygon description that is in some sense a "best fit" to the sensor points the model segment represents. A set of sensor points is used to update the surface model by creating a point segment for each of the sensor points. The creation of these point segments may result in the formation of gaps in the surface model, so various model segment operations are performed in an attempt to fill these gaps, and reduce the complexity of the surface model:

- Two model segments are merged to form a new model segment by merging their model segment data and their descriptions. Depending on the types

of model segments being merged, different algorithms are used to merge the model segments' descriptions:

  - If two point segments are being merged then the new model segment's description is the point or planar polygon of best fit.
  - If a point segment and planar polygon segment are being merged then their descriptions are projected onto the plane of the new model segment, and a perimeter section of the projected planar polygon is extended to the projected point.
  - If two planar polygon segments are being merged then their descriptions are projected onto the plane of the new model segment. If the projections do not overlap then a perimeter section of one of the projected planar polygons is extended to a perimeter section of the other projected planar polygon. If the projections overlap then the outermost perimeter is constructed and any holes' perimeters are collapsed.

  Any shared edges inherited by planar polygon segments resulting from mergers are then correctly repositioned.

- Fissures in a planar polygon segment's perimeter are filled. Filling one fissure may result in the formation and subsequent filling of another fissure, and so on.

- Shared edges are formed between two planar polygon segments by determining the concavity and intersection line between the model segments' planes. Then a perimeter section from each model segment is extended to a common section of the intersection line.

- A corner vertex is formed between successive shared edges by determining the point of intersection of the intersection lines along which the shared edges lie. Then the perimeter section that lies between the shared edges is extended to the intersection point. If a corner vertex is formed then its corresponding corner vertices on other planar polygon segments must also be formed.

During these model segment operations, shared edges may be unshared, and redundant model segments may be deleted.

Means of collecting the sets of sensor points via a robot are required. This is called surface following. The robot's surface following motion must be controlled such that the sensors can take surface readings, and the robot avoids collision with the surface. Also as much of the surface as possible must be exposed to the robot's sensors. Chapter 3 presents means of computing the robot's surface following motion.

Chapter 3

Surface Following

3.1 Introduction

Chapter 2 presented techniques for incrementally constructing a surface model from sets of sensor points. These sensor points are obtained from range sensors mounted on a robot that moves the sensors over the surface being modelled. This robot motion is called surface following, and this chapter presents means of computing a robot's surface following motion.

The range sensors are mounted around the robot's control point. The robot must move its control point over the surface closely enough for the range sensors to take surface readings but not so closely that the control point collides with the surface. To achieve this, the control point follows the surface model while maintaining a desired distance from the model. While the control point is following the surface model the rest of the robot must be prevented from colliding with the surface. To achieve this, redundant degrees-of-freedom are used to keep the rest of the robot away from the surface model.

The aim of surface modelling and surface following is to model as much of the surface as possible. To achieve this, the robot's range sensors must be exposed to as much of the surface as possible. The surface model's non-shared edges indicate where sections of unexposed surface lie, so surface following directions are determined that direct the robot's control point to non-shared edges. Once the control

point is at a non-shared edge, surface following directions are determined that direct the control point over the unexposed surface near the edge. A surface following direction forms a component of each control point motion.

The layout of this chapter is as follows: In Section 3.2 motion of the robot's control point is considered. Section 3.3 describes the use of redundancy to prevent the rest of the robot from colliding with the surface. Section 3.4 presents the algorithm that determines surface following directions, and the chapter concludes with a summary in Section 3.5.

3.2 Control Point Motion

The robot must move its control point over the surface closely enough for the range sensors to take surface readings but not so closely that the control point collides with the surface. To achieve this, the control point follows the surface model while maintaining a *desired distance* δ from the model. By setting δ less than the sensors' sensing range, the control point's sensors are kept within sensing range of the surface. As the desired distance is maintained between the control point and the surface model, not the actual surface, δ is set greater than the surface model's linear resolution. This prevents the control point from colliding with the surface.

To maintain the desired distance between the control point and the surface model, each control point motion is the vector sum of a *corrective motion* and a *perpendicular motion*, as shown in Figure 27. The corrective motion moves the control point to the desired distance from the surface model, and the perpendicular motion moves the control point along the surface model. To compute the corrective motion, the surface model point closest to the control point is determined. The corrective motion's direction is the vector from the closest surface model point to the control point, and the corrective motion's length is the desired distance minus the distance of the control point from the closest surface model point, as shown in Figure 27.

The perpendicular motion's direction is obtained by orthogonally projecting the surface following direction onto the plane that is perpendicular to the corrective

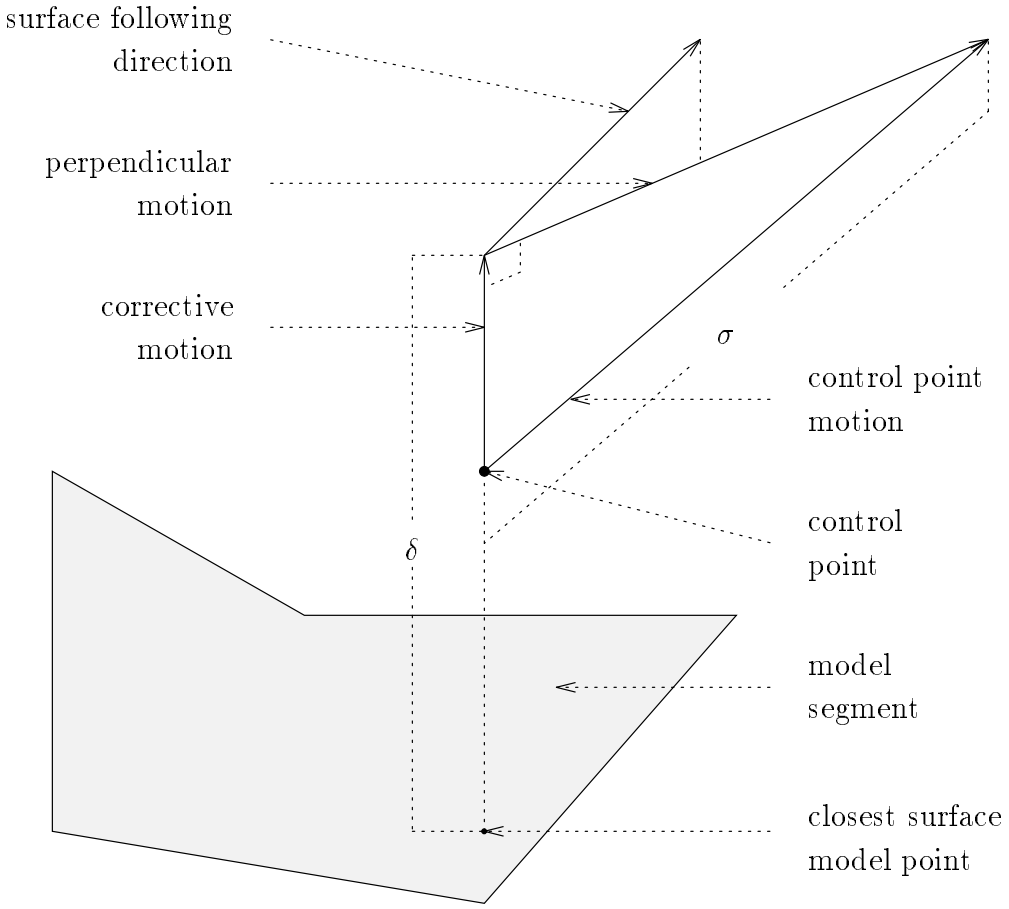


Figure 27: The control point motion.

motion. Calculation of the surface following direction is described in Section 3.4. The perpendicular motion's length is chosen such that the control point motion's length equals a set *step size* σ, as shown in Figure 27. The corrective motion is given priority over the perpendicular motion such that if the corrective motion's length is greater than σ then the corrective motion's length becomes σ, and the perpendicular motion's length becomes zero.

Normally, σ is set to a fixed maximum value $\sigma_{\max}$. However, σ is sometimes decreased, as described in Section 3.4. Also σ is never allowed to be greater than the distance between the control point and the closest surface model point. This prevents a control point motion from moving the control point into the surface model. For safety's sake each control point motion must be prevented from moving the control point out of the region sensed by the control point's sensors. This is

achieved by setting $\sigma_{\max}$ less than the sensors' sensing range. Also the step size affects the distance between one set of sensor points and the next, so $\sigma_{\max}$ is set such that the next set of sensor points overlaps the previous set.

Once the control point motion has been determined, the actuator changes that achieve this motion must be computed. Given the forward kinematics equations that specify the control point's position $\mathbf{x}$ in terms of the actuators' positions Θ, small changes in the control point's position can be related to small actuator changes by the Jacobian of the forward kinematics equations [71,80,99]:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\Theta}, \tag{2}$$

The actuator changes that achieve a desired control point motion $\dot{\mathbf{x}}$ can be determined by solving (2):

$$\dot{\Theta} = \mathbf{G}\dot{\mathbf{x}},$$

where $\mathbf{G}$ is a generalized inverse of $\mathbf{J}$. Note that $\mathbf{G} = \mathbf{J}^{-1}$ if the robot is not kinematically redundant.

Thus far only motion of the robot's control point has been considered. The rest of the robot must also be prevented from colliding with the surface. If δ is much greater than the distance between the control point and all other points on the robot then maintaining δ between the control point and the surface model will prevent the entire robot from colliding with the surface. However, this is seldom the case, especially for manipulator type robots.

3.3 Using Redundancy for Collision Avoidance

The control point motion is a three-dimensional vector, so to move the control point the robot requires at least three degrees-of-freedom. For the task of moving the control point, robots with greater than three degrees-of-freedom are known as *kinematically redundant* robots. In surface following, redundant degrees-of-freedom are used to keep the rest of the robot away from the surface model.

For a redundant robot there is an infinite number of solutions to (2). A particular solution must be chosen that keeps the rest of the robot away from the surface

model. Liégeois [64] and Klein and Huang [60] showed how homogeneous solutions to (2) can be used to optimize some objective function in addition to moving the control point:

$$\dot{\Theta} = \mathbf{G}\dot{\mathbf{x}} + \alpha(\mathbf{I}_n - \mathbf{G}\mathbf{J})\mathbf{z}, \tag{3}$$

where α is a fixed gain value, and $\mathbf{z}$ is an n-vector; n being the number of degrees-of-freedom of the robot. The first term of (3) provides a set of actuator changes that achieve the desired control point motion. The second term of (3) uses a projection operator to select a component of $\mathbf{z}$ from the space of homogeneous solutions to (2). Thus the second term reconfigures the rest of the robot without affecting the position of the control point. Combining these two terms provides a set of actuator changes that perform the desired control point motion and reconfigure the rest of the robot.

Usually, $\mathbf{J}^+$ the Moore-Penrose pseudoinverse [88] of $\mathbf{J}$ is used for $\mathbf{G}$. The pseudoinverse provides a minimum norm solution in the first term of (3), and in the second term orthogonally projects $\mathbf{z}$ onto the null-space of $\mathbf{J}$. Usually, $\mathbf{z}$ is obtained from the gradient of some objective function $\mathcal{H}$ to be optimized.

In surface following the objective is to keep the rest of the robot away from the surface model. To achieve this, an objective function was developed that is based on the surface model's distance from each part (link) of the robot. For each link i, the link point $\mathbf{l}_i$ that is closest to the surface model, and the corresponding surface model point $\mathbf{s}_i$ that is closest to link i are determined. To obtain these points, a description of each of the robot's links is required. The objective function to be optimized is given by:

$$\begin{aligned} \mathcal{H}(\Theta) &= \frac{d_0}{g}\sum_{i=1}^{n}\begin{cases} (1-\frac{d_i}{d_0})^g & d_i < d_0 \\ 0 & d_i \geq d_0 \end{cases}, \\ d_i &= \frac{1}{2}\|\mathbf{l}_i - \mathbf{s}_i\|^2 = \frac{1}{2}(\mathbf{l}_i - \mathbf{s}_i).(\mathbf{l}_i - \mathbf{s}_i), \end{aligned} \tag{4}$$

where g is a positive integer, and $\sqrt{2d_0}$ is the distance from the surface model beyond which a link has no influence on the objective function. Setting $\sqrt{2d_0}$ equal to the sensing range of the robot's sensor usually gives good results. Note that each $\mathbf{l}_i$ is a function of actuator positions $\theta_1, \ldots, \theta_i$. The objective function is zero

if all of the robot's links are further than $\sqrt{2d_0}$ from the surface model, that is, if $d_i \geq d_0$ for all i. If the ith link is within this distance, that is, if $d_i < d_0$ then $\mathcal{H}(\Theta)$ is positive and increases as the link gets closer to the surface model. If the link intersects the surface model then $(1 - d_i/d_0)^g = 1$. Thus the maximum value of the objective function is nd_0/g.

The reconfiguration vector is obtained from the negative gradient of the objective function:

$$\mathbf{z} = - \bigtriangledown \mathcal{H}(\Theta).$$

Thus the components of the reconfiguration vector are given by:

$$z_k = \sum_{i=k}^{n} \begin{cases} (1 - \frac{d_i}{d_0})^{g-1} (\mathbf{l}_i - \mathbf{s}_i).\mathbf{r}_{ik} & d_i < d_0 \\ 0 & d_i \geq d_0 \end{cases}, \tag{5}$$

$$\mathbf{r}_{ik} = \begin{cases} \hat{\mathbf{a}}_k \times (\mathbf{l}_i - \mathbf{q}_k) & k\text{th joint revolute} \\ \hat{\mathbf{a}}_k & k\text{th joint prismatic} \end{cases},$$

where $\hat{\mathbf{a}}_k$ is the unit vector that describes the kth actuator's axis of motion, and $\mathbf{q}_k$ is the kth actuator's position in space. Note that $\mathbf{l}_i, \mathbf{s}_i, \hat{\mathbf{a}}_k$ and $\mathbf{q}_k$ are defined relative to a global coordinate frame. A full derivation of (5) is given in Appendix A.

It is often the case that the closest link point of the control point's link is close to or coincides with the control point. In such cases, generating a reconfiguration vector for this link is unnecessary since the control point is not moved by reconfiguration. Therefore the description of the control point's link does not include the control point and the region of radius δ surrounding the control point.

In some configurations, no suitable reconfiguration vector exists. For example, if the links of the robot are wrapped around a circular object. Thus the robot can become stuck in such configurations. The surface following algorithm detects these situations and takes appropriate action, as will be described in Section 3.4.7.

As with the control point, each motion of the robot must be prevented from moving any part of the rest of the robot out of the region sensed by the robot's sensors. This is achieved by placing a fixed ceiling on the length of the vector of actuator changes computed by (3). If $\|\dot{\Theta}\|$ is greater than a ceiling value γ then $\dot{\Theta}$ is scaled to have length γ. The value of γ is set such that no actuator changes

satisfying $\|\dot{\Theta}\| < \gamma$ will move any part of the robot a distance greater than the sensing range of the robot's sensors. If the robot has both prismatic and revolute joints then $\dot{\Theta}$ is partitioned into its prismatic and revolute components, and there are separate ceilings γ_{p} and γ_{r}, respectively, for each type of joint.

Determining γ for prismatic joints is straightforward because the relationship between the motion of prismatic joints and the control point is linear. For revolute joints, γ can usually be determined by configuring the robot such that the distance between any revolute joint axis and any point on the robot is maximized. Let $\mathbf{j}$ be the joint and $\mathbf{x}$ be the point on the robot that achieve this maximum distance r. With the robot configured such that the distance between joint axis $\mathbf{j}$ and point $\mathbf{x}$ is r, rotation of joint $\mathbf{j}$ by an amount θ will move $\mathbf{x}$ a distance $r\theta$, and as r is maximal this is the greatest distance any point on the robot can be moved by any joint rotation θ. Therefore, to ensure that all points on the robot are moved a distance less than the sensing range of the robot's sensors, for revolute joints γ is set less than the sensing range divided by r.

The objective function (4) uses the distances of the links from the surface model to generate a reconfiguration vector. Thus for a link to be reconfigured away from part of the surface, that part of the surface must be modelled. This means that the links must be covered with range sensors. If a link is within sensing range of the surface then its range sensors will detect sensor points. These sensor points will be incorporated into the surface model, and the surface model will then cause the link to be reconfigured away from the surface.

The cost of covering the links with range sensors may be prohibitively expensive. Alternatively, the links may be covered with simple proximity switches that detect the presence of an object that is within the switch's sensing field. Cheung and Lumelsky [20] have developed a robot "skin" that consists of an array of infra-red emitter/receiver pairs mounted on a flexible circuit board that can be wrapped around robot links to provide sensor coverage. Proximity switches provide a binary signal: object present/no object present, and so their data cannot be used for surface modelling. If the links are covered with proximity switches then a different objective function is required. For each proximity switch that detects the presense

of the surface, a reconfiguration vector should be generated that reconfigures the switch's link away from the surface. An objective function was developed for use with proximity switches, and is given in Appendix B.

The use of redundant degrees-of-freedom is a popular approach to collision avoidance [10,31,53,58,67]. Redundancy has also been used to achieve a variety of other objectives, such as minimising kinetic energy [99,54], maintaining dexterity [22,59], and avoiding joint limits [22,60,62,64]. Within the context of surface following, redundancy has also been used to avoid joint limits [84].

3.4 Surface Following Directions

The aim of surface modelling and surface following is to model as much of the surface as possible. To achieve this, the robot's range sensors must be exposed to as much of the surface as possible. The surface model's non-shared edges represent the boundary between sections of surface that have been exposed to the robot's sensors and sections of surface that have not. Thus the non-shared edges indicate where sections of unexposed surface lie. *Surface following directions* are determined that direct the robot's control point to non-shared edges. Once the control point is at a non-shared edge, surface following directions are determined that direct it over the unexposed surface near the edge. This is intended to result in sensor points being detected on the unexposed surface so that surface modelling can extend the surface model into the unexposed region. As mentioned in Section 3.2, a surface following direction forms a component of each control point motion. This section describes the algorithm that determines surface following directions.

When called for the first time, the algorithm chooses a *current surface*. The current surface is an incomplete surface that is within sensing range of the control point's sensors. Recall from Section 2.2, that a surface is a set of model segments connected by the neighbour relation. A surface is complete if all of its model segments are complete. A model segment is complete if all of its edges are complete, or if it is marked unreachable. An edge is complete if it is a shared edge, or if it is marked unreachable or unclosable. The control point is directed to the current

surface's incomplete edges by choosing a *current segment*, and directing the control point to it. The current segment is an incomplete model segment of the current surface, so its perimeter must contain incomplete edges. If the control point cannot reach the current segment then the current segment is marked unreachable, another current segment is chosen, and the control point directed to it. Once the control point reaches the current segment, a *current edge* is chosen, and the control point directed to it. If the control point cannot reach the current edge then the current edge is marked unreachable, and another current edge is chosen. Once the control point reaches the current edge, it is directed over the unexposed surface near the edge. If surface modelling changes the current edge then another current edge is chosen, and the control point directed to it. Surface modelling changing the current edge does not necessarily imply that a shared edge has been formed. However, it does mean that the perimeter of the current segment has been extended, and this is the next best thing. Section 3.4.6 describes what is meant by surface modelling changing an edge. If the control point has been directed over much of the unexposed surface near the current edge without surface modelling changing the current edge then the current edge is marked unclosable, another current edge is chosen, and the control point directed to it. Once all of the current segment's edges are complete, the current segment must also be complete, and so another current segment is chosen, and the control point directed to it. Once all of the current surface's model segments are complete, the current surface must also be complete, and so surface modelling and surface following stop, and the robot is halted. Figure 28 shows a schematic description of the algorithm.

In some cases the completed surface model may contain unreachable model segments and/or unclosable or unreachable edges. These model segments and edges may be reachable or closable via motions of the control point not attempted by the surface following algorithm. Usually only one attempt is made to reach or close each model segment or edge. If this fails then the model segment or edge is marked unreachable or unclosable. However, results have shown that one attempt is usually sufficient to reach or close a model segment or edge and so completion of the current surface usually results in the modelling of as much of the surface as possible.

```
current surface = incomplete surface within sensing range of control point;
while (current surface incomplete)
{
  current segment = incomplete model segment of current surface;
  direct control point to current segment;
  if (cannot reach current segment)
    mark current segment unreachable;
  else
    while (current segment incomplete)
    {
      current edge = incomplete edge of current segment;
      direct control point to current edge;
      if (cannot reach current edge)
        mark current edge unreachable;
      else
        while (current edge not changed && current edge incomplete)
        {
          direct control point over unexposed surface near current edge;
          if (have moved over much of unexposed surface near current edge)
            mark current edge unclosable;
        }
    }
}
```

Figure 28: A schematic description of the surface following algorithm.

The algorithm computes surface following directions in one of five different motion modes:

Current surface following: The control point is directed to the current segment via a path along the current surface.

Current segment following: The control point is directed to the current edge via the perimeter of the current segment.

Current edge following: The control point is directed over the unexposed surface near the current edge. This is intended to result in sensor points being detected on the unexposed surface so that surface modelling can extend the surface model into the unexposed region.

Point following: A special case of current edge following, where the current segment is a point segment rather than a planar polygon segment. The control point is directed over the unexposed surface that surrounds the current segment. This is intended to result in sensor points being detected near the current segment so that surface modelling can transform the current segment into a planar polygon segment.

Gap following: If during current edge following or point following, a gap in the surface model is detected near the control point then gap following directs the control point's sensors over the gap. This is intended to result in sensor points being detected in the gap so that surface modelling can close the gap.

The first two motion modes direct the control point to sections of unexposed surface, and the last three motion modes may be thought of as exploratory procedures that direct the control point over unexposed sections of the surface. The motion modes are described in detail in the remainder of this section.

3.4.1 Current Surface Following

Current surface following directs the control point to the current segment via a path along the current surface. The path consists of a set of connected edges on the current surface. Current surface following determines the path and then directs the control point along it. A branch-and-bound search of the current surface's edges is used to determine the path. The search starts from the current surface edge that is nearest the control point, and the goal of the search is to find a path to an incomplete model segment. The branch-and-bound search yields the shortest such edge path. The incomplete model segment at the end of the path becomes the current segment. If the current surface has no incomplete model segments then the current surface is complete, and so surface following and surface modelling stop.

Once a path has been found, the control point is directed along it by directing the control point along the path's edges. The path edge that the control point is currently following is called the *follow edge*. Initially, the follow edge is the first edge of the path. Thereafter, it is chosen from the path edges that lie between the

previous follow edge and final path edge, inclusive. Of these edges, the follow edge becomes the edge that is furthest along the path and a distance less than ρ from the control point, where ρ is the furthest the control point should be allowed to stray from the path. If none of these edges is within ρ of control point then the follow edge remains unchanged. Thus the choice of follow edge cannot move backwards along the path. Good results have been obtained by setting ρ to the sensing range of the robot's sensors.

If surface modelling changes any path edge, from the follow edge onwards, or the current segment becomes complete then current surface following restarts. This results in a new path being determined. Recomputation of a new path due to surface modelling changes seldom occurs in practice because path edges are usually shared edges, and shared edges are seldom changed by surface modelling, that is, once shared they usually remain shared edges until surface modelling and surface following complete. Non-shared path edges are more likely to be changed by surface following but occur less frequently. Also most sensor points are obtained from the sensors around the control point, so most surface modelling changes will occur near the control point. Thus if a path edge is changed it is likely to be near the control point and so the recomputed path is likely to be short and thus found quickly.

While being directed along the path, the control point is made to stay close to the path. This is necessary for the control point to reach the current segment. To achieve this, the surface following direction $\mathbf{s}$ is the sum of two vectors:

- a vector $\mathbf{v}_1$ from the control point to the follow edge point that is nearest the control point, and
- a vector $\mathbf{v}_2$ that is parallel to the follow edge, and in the direction of the path.

The first vector attracts the control point towards the path, and the second moves the control point along the path, as shown Figure 29(a). The ratio with which the two vectors are combined depends upon the distance $\|\mathbf{v}_1\|$ of the control point from the follow edge:

$$\mathbf{s} = r\frac{\mathbf{v}_1}{\|\mathbf{v}_1\|} + (1-r)\frac{\mathbf{v}_2}{\|\mathbf{v}_2\|} \tag{6}$$

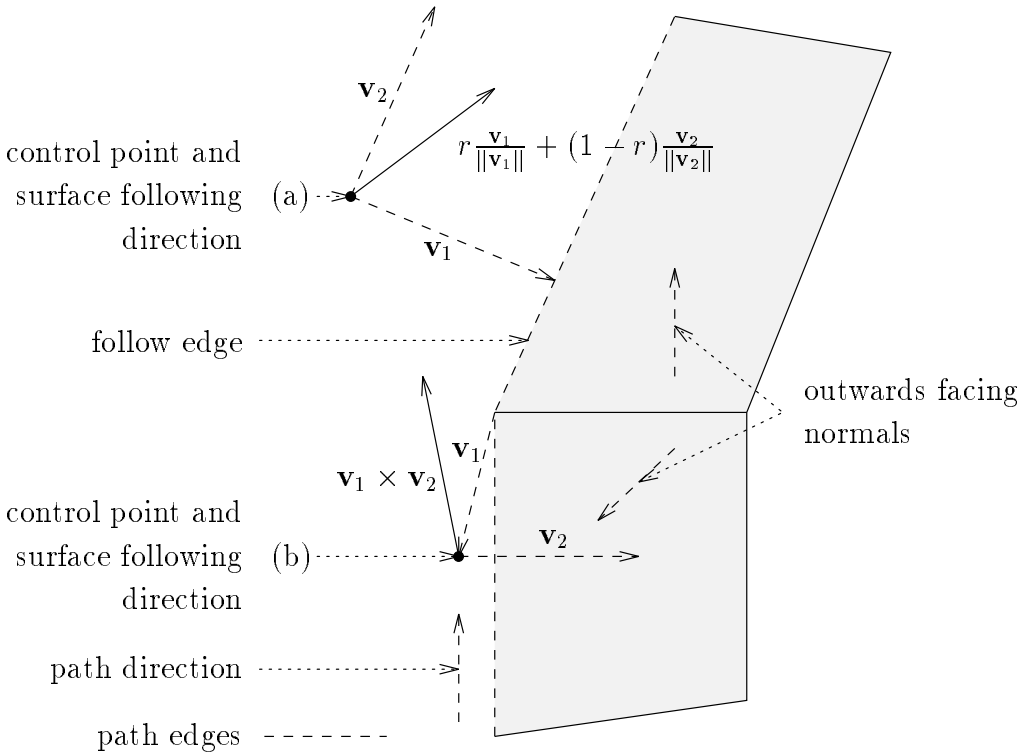


Figure 29: Current surface following. (a) The control point is directed towards and along the path. (b) The control point is directed towards the outside of a model segment's plane.

$$r = \begin{cases} 0 & \|\mathbf{v}_1\| \leq \delta \\ \frac{\|\mathbf{v}_1\|-\delta}{\rho-\delta} & \delta < \|\mathbf{v}_1\| < \rho \\ 1 & \|\mathbf{v}_1\| \geq \rho \end{cases} .$$

The further the control point is from the follow edge, the more the surface following direction is directed towards the follow edge. If the distance of the control point from the follow edge is greater than ρ then the surface following direction is directly towards the follow edge. The closer the control point is to the follow edge, the more the surface following direction is directed parallel to the follow edge and in the direction of the path. If the control point is within the desired distance of the follow edge then the surface following direction is parallel to the follow edge and in the direction of the path.

The control point is made to remain on the outside of the planes of any model segments to which the follow edge belongs. This prevents the control point from

being directed underneath such model segments. If the follow edge is a shared edge then it belongs to two neighbouring model segments. If the control point is on the inside of the plane of a model segment to which the follow edge belongs then the control point must be directed towards the outside of the plane. To achieve this, the model segment's edge that is closest to the control point is determined. The surface following direction is then the cross product of two vectors (refer to Figure 29(b)):

- a vector $\mathbf{v}_1$ from the closest edge to the control point, and
- a vector $\mathbf{v}_2$ that is parallel to the closest edge, and in an anticlockwise direction along the closest edge.

Once the control point reaches the current segment, the choice of current edge is cleared and the motion mode switches to current segment following. Clearing the choice of current edge results in a new current edge being chosen, as described in Section 3.4.2. If the control point cannot reach the current segment then the current segment is marked unreachable, and current surface following restarts. This results in a path to a new current segment being determined. The current segment being marked unreachable does not mean that it is impossible for the control point to be moved to the current segment; it means that the control point could not follow the path to the current segment. It may be possible for the control point to be moved to the current segment via some other path. However, it would be too time consuming to repeatedly test paths until one is found that the control point can follow to the current segment. So if the control point cannot follow the shortest edge path to the current segment then the current segment is marked unreachable. If a model segment is marked unreachable and the control point is subsequently brought closest to the model segment then the model segment is re-marked incomplete. The implications of marking model segments unreachable, for the final surface model and termination of surface following are discussed in Section 3.4.7.

3.4.2 Current Segment Following

Current segment following directs the control point to the current edge via the current segment's perimeter. Instead of following the perimeter a simpler solution

would be to direct the control point in a straight line towards the current edge. However, a model segment's perimeter may be concave, so a straight line path between two model segment points may pass outside of the model segment's perimeter. Directing the control point along such a straight line could make it difficult for the control point to maintain the desired distance from the surface model.

Current segment following first checks whether the choice of current edge has been cleared, or whether the current edge has been completed. The choice of current edge is cleared whenever surface modelling changes the current edge. It is also cleared at various points in the algorithm such as when the motion mode switches from current surface following to current segment following, as described in Section 3.4.1. If the choice of current edge has been cleared, or the current edge has been completed then a new current edge is chosen. Since current segment following directs the control point along the current segment's perimeter, the current segment edge f that is closest to the control point is first determined. The new current edge then becomes the current segment's incomplete edge that is closest to f, via the current segment's perimeter. If there are no incomplete edges on the current segment then the current segment is complete, and so the motion mode switches to current surface following, where a path to a new current segment is determined.

Occasionally surface modelling forms zero length edges. If the current edge has zero length then it is marked unclosable, and the current segment following procedure restarts. This results in a new current edge being chosen.

Once a current edge has been chosen, the control point is directed to the current edge via the current segment's perimeter. The current segment edge that the control point is currently following is called the *follow edge*[1]. The direction (clockwise/anticlockwise) in which the control point is directed along the current segment's perimeter, is called the *current direction*. When a new current edge is chosen, the follow edge becomes f, and the current direction becomes the direction that yields the shorter of the clockwise and anticlockwise routes from f to the current edge. Thereafter, the follow edge is chosen from the current segment's perimeter section that lies in the current direction between the previous follow edge

[1]Similar to the follow edge used in current segment following.

and current edge, inclusive. Of these edges, the follow edge becomes the edge that is furthest along the perimeter section (in the current direction) and a distance less than ρ from the control point. If none of these edges is within ρ of control point then the follow edge remains unchanged. Thus the choice of follow edge cannot move against the current direction. As with the current edge, current surface following also checks whether the choice of follow edge has been cleared. The choice of follow edge is cleared whenever surface modelling changes the follow edge. It is also cleared at various points in the algorithm. If the choice of follow edge has been cleared then a new follow edge and current direction are chosen. The new follow edge is the current segment's edge that is nearest the control point, and the new current direction is the direction that yields the shorter of the clockwise and anticlockwise routes from the follow edge to the current edge.

The surface following directions for current segment following are analogous to those of current surface following, as described in Section 3.4.1. While being directed along the perimeter, the control point is made to stay close to the perimeter. This is necessary for the control point to reach the current edge. To achieve this, the surface following direction $\mathbf{s}$ is the sum of two vectors:

- a vector $\mathbf{v}_1$ from the control point to the follow edge point that is nearest the control point, and
- a vector $\mathbf{v}_2$ that is parallel to the follow edge, and in the current direction.

The first vector attracts the control point towards the current segment's perimeter, and the second moves the control point along the perimeter, as shown in Figure 30(a). The ratio with which the two vectors are combined depends upon the distance $\|\mathbf{v}_1\|$ of the control point from the follow edge as in (6). The further the control point is from the follow edge, the more the surface following direction is directed towards the follow edge. If the distance of the control point from the follow edge is greater than ρ then the surface following direction is directly towards the follow edge. The closer the control point is to the follow edge, the more the surface following direction is directed parallel to the follow edge, and in the current direction. If the control point is within the desired distance of the follow edge then

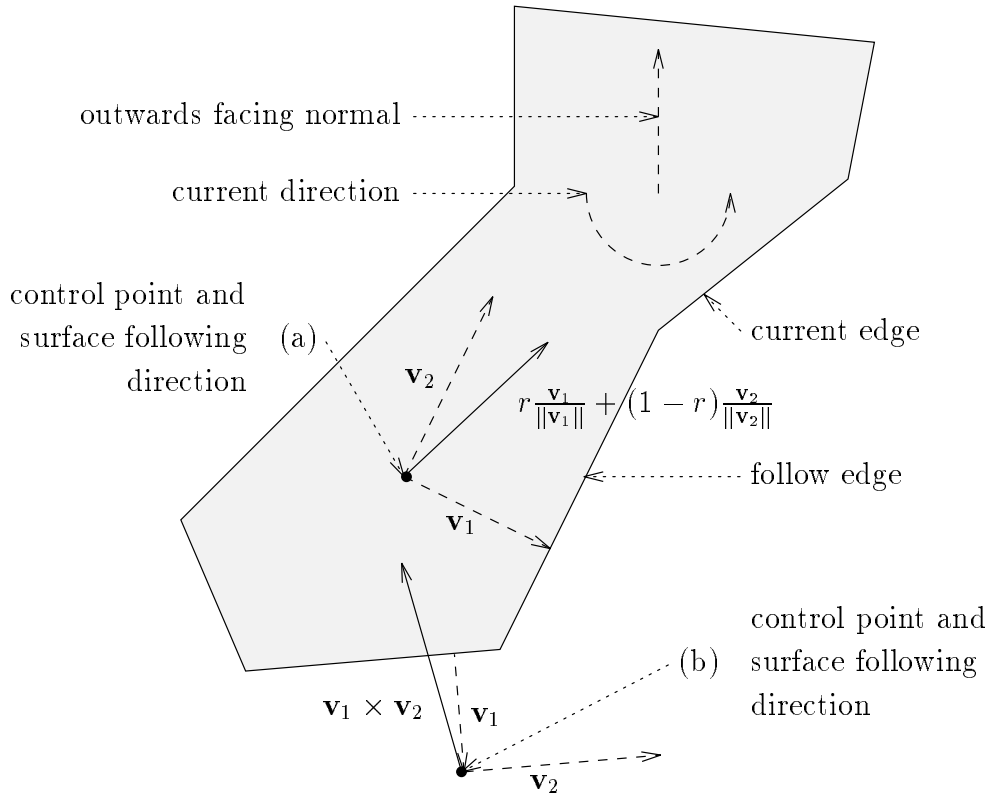


Figure 30: Current segment following. (a) The control point is directed towards and along the follow edge. (b) The control point is directed towards the outside of the current segment's plane.

the surface following direction is parallel to the follow edge, and in the current direction.

The control point is made to remain on the outside of the current segment's plane. This prevents the control point from being directed underneath the current segment. If the control point is on the inside of the current segment's plane then it must be directed towards the outside of the plane. To achieve this, the current segment's edge that is closest to the control point is determined. The surface following direction is then the cross product of two vectors (refer to Figure 30(b)):

- a vector $\mathbf{v}_1$ from the closest edge to the control point, and
- a vector $\mathbf{v}_2$ that is parallel to the closest edge, and in an anticlockwise direction along the closest edge.

Once the control point reaches the current edge, the motion mode switches to

current edge following. If the control point cannot reach the current edge then the current edge is marked unreachable, and current segment following restarts. This results in a new current edge being chosen. The current edge being marked unreachable does not mean that it is impossible for the control point to be moved to the current edge; it means that the control point could not follow the current segment's perimeter to the current edge. It may be possible for the control point to be moved to the current edge via some other path. However, it would be too time consuming to repeatedly test paths until one is found that the control point can follow to the current edge. So if the control point cannot follow the current segment's perimeter to the current edge then the current edge is marked unreachable. If an edge is marked unreachable and the control point is subsequently brought closest to the edge then the edge is re-marked incomplete. The implications of marking edges unreachable, for the final surface model and termination of surface following are discussed in Section 3.4.7.

In some cases it is just as good to direct the control point to any incomplete edge as it is to direct the control point to the current edge. Conversely, it is sometimes important that the control point be directed to the current edge in particular. These two cases are described further in Section 3.4.3. The former case is indicated by a flag being set, and the latter by the flag being cleared. Thus if the flag is set and the follow edge is incomplete then the current edge becomes the follow edge and the motion mode switches to current edge following. Initially the flag is set, thereafter it is set and cleared during current edge following, as will be described in Section 3.4.3.

3.4.3 Current Edge Following

Current edge following directs the control point over the unexposed surface near the current edge. This is intended to result in sensor points being detected on the unexposed surface so that surface modelling can extend the surface model into the unexposed region. The unexposed surface near the current edge is divided into *slices* and the control point is directed along each slice in turn. The slices are defined relative to a *slice line*, as shown in Figure 31. The slice line passes through the *slice*

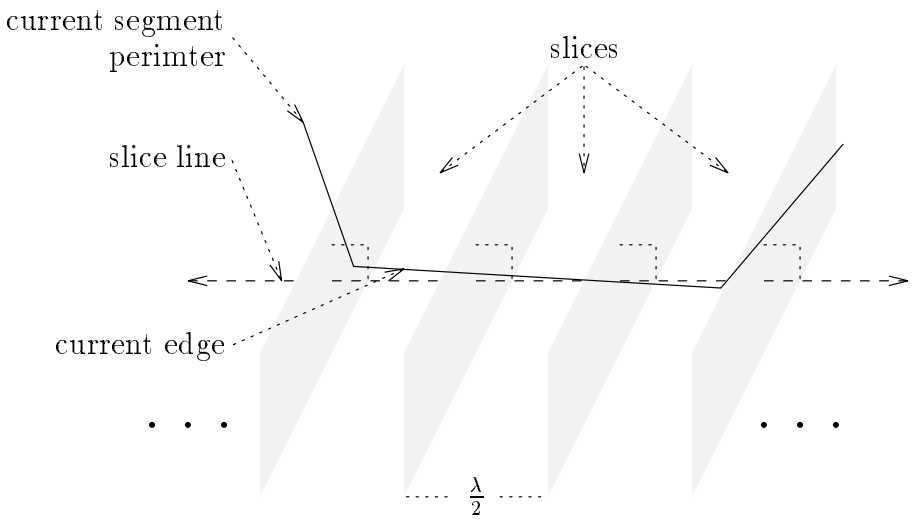


Figure 31: The slice line and slices.

point and is parallel to the *slice vector*; these will be defined later. Each slice is half as wide as the surface model's linear resolution, and perpendicular to the slice line. The slice line may not necessarily coincide with the current edge but is close to it.

A slice passing through the current edge is chosen, and the control point is directed along it. This slice is called the *current slice*. Directing the control point along the current slice is intended to result in sensor points being detected on the unexposed surface within the current slice. However, this is unlikely if there is only a small area of unexposed surface within the current slice. In such cases it is wasteful to direct the control point along the current slice. So whenever a new current slice is chosen, it is checked as follows: The current edge point nearest the centre of the current slice is determined. This point is called the *current point*. If part of the surface model lies *outside* of the current edge, and is a distance less than half the surface model's linear resolution from the current point then it is likely that there is only a small area of unexposed surface within the current slice. In such cases the current slice is deemed used, and a new current slice is chosen. A surface model point lies outside the current edge if either of the following is satisfied:

- The point does not lie on the current segment.
- The point lies on the current segment, as shown in Figure 32, is strictly

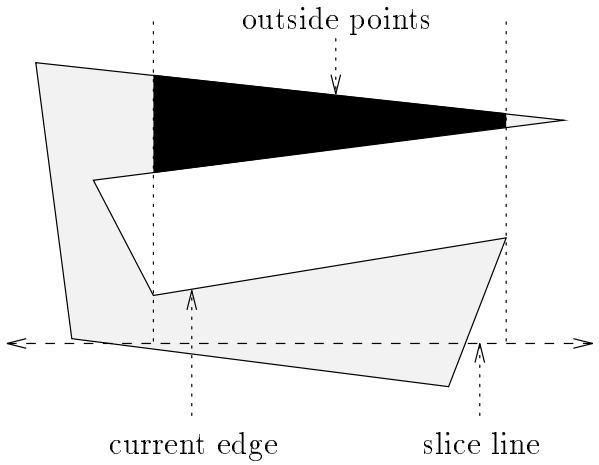


Figure 32: Points on the current segment that lie outside of the current edge.

between the lines that are perpendicular to the slice line and pass through the current edge's vertices, and any vector **q** from the current edge to the point, points away from the current edge, that is:

$$(\mathbf{e} \times \mathbf{n}).\mathbf{q} > 0,$$

where **e** is a vector that is parallel to the current edge and in an anticlockwise direction along the current edge, and **n** is the current segment's outwards facing normal.

When the current edge is chosen, the current slice is also chosen. Thereafter, whenever current segment following or current edge following is called, the current slice is checked as described above.

The current slice is chosen from the unused slices that pass through the current edge, and are immediately adjacent to a used slice. If there are no used slices then any unused slice passing through the current edge is chosen. Thus as slices are used they form a block of adjacent used slices. Once the block of used slices contains the entire current edge, the current edge is marked unclosable, and the motion mode switches to current segment following, where a new current edge is chosen.

A slice may be marked used for the following reasons:

- The slice contains only a small area of unexposed surface.

- A gap detected in the slice could not be filled. This will be discussed further in Section 3.4.5.

- Current edge following made insufficient progress towards or along the slice. This will be discussed further in Section 3.4.7.

Thus if the block of used slices contains the current edge then an attempt has been made to move the control point over any substantial area of unexposed surface near the edge. This also signifies that the current edge has not been changed significantly by surface modelling, as will be described in the following paragraphs, and so the current edge is marked unclosable, that is, the attempts to move the control point over the unexposed surface near the current edge did not result in surface modelling significantly changing it.

Each time current edge following is called, it first checks whether the choice of current edge has been cleared. If so then the motion mode switches to current segment following, where a new current edge is chosen. Each time a new current edge is chosen, the slice line is tested to see whether the slices it defines can be used with the new current edge. The slices are retained for use with the new current edge if the following are satisfied (refer to Figure 33):

- The previous current edge belonged to the current segment.

- The distance of the current edge's vertices from the slice line is less than the surface model's linear resolution.

- The angle between the slice vector and a vector that is parallel to the current edge, and in an anticlockwise direction along the current edge, is less than 45°.

- At least one used slice passes through the current edge.

If the slices are not retained then a new slice line is defined by choosing a new slice point and slice vector. The new slice point is any point on the new current edge, and the new slice vector is parallel to the current edge, and in an anticlockwise direction along the current edge. The flag examined by current segment following

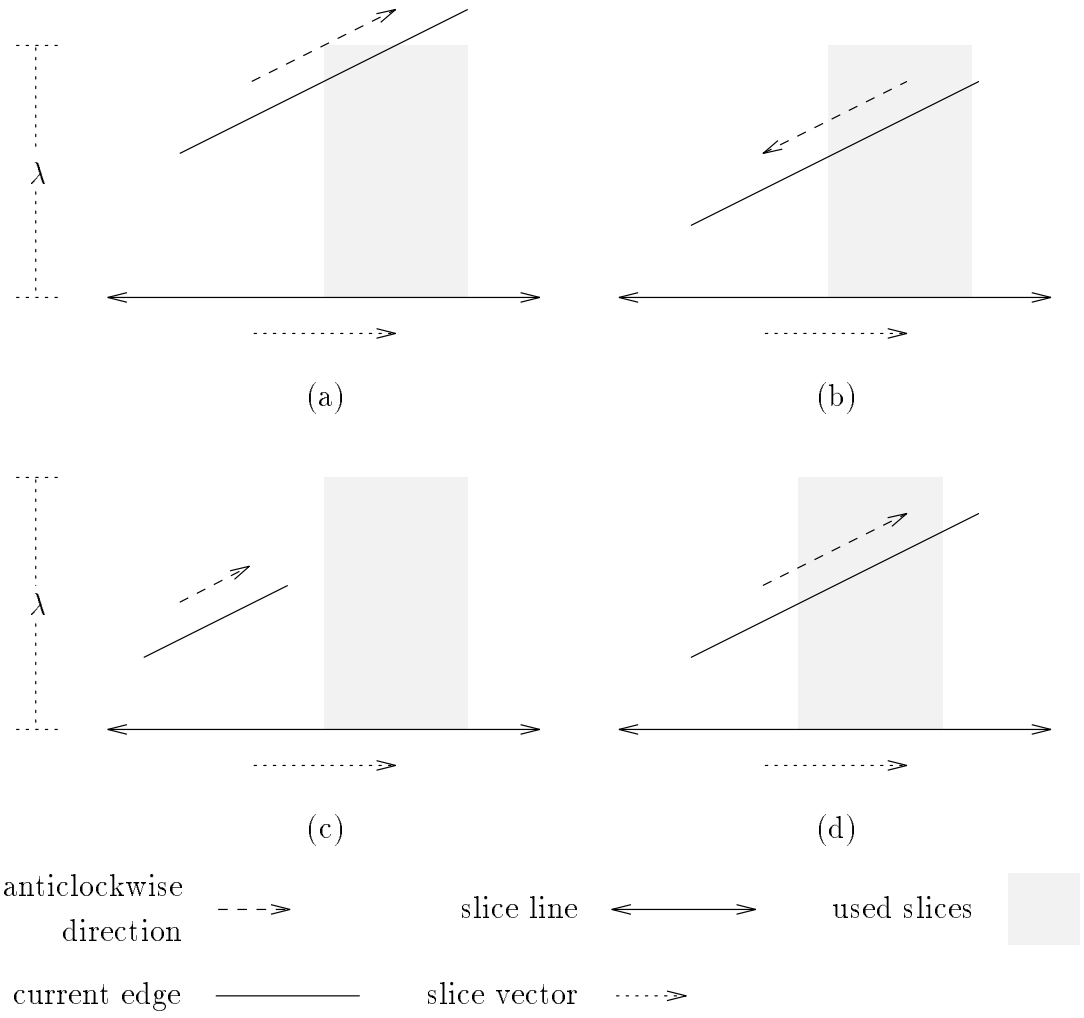


Figure 33: Retaining the slice line. (a) The slice line is not retained as a vertex is too far from the slice line. (b) The slice line is not retained as the angle is too great. (c) The slice line is not retained as no used slices pass through the current edge. (d) The slice line is retained.

when the follow edge is incomplete, is set whenever a new slice line is defined. It is cleared when the current slice is deemed used. Thus if there are no used slices then the flag is set, and so current segment following sets the current edge to any incomplete follow edges, and the motion mode switches to current edge following. If there are used slices then the flag is cleared, and so current segment following ignores incomplete follow edges, and directs the control point to the current edge.

Rather than defining slices relative to the slice line, they could be defined relative to the current edge. However, when surface modelling forms a new section of

perimeter, the new section is often quite close to the section that it replaces. Thus the current edge is often quite close to previous current edges. Therefore if slices were defined relative to the current edge then they would often be similar to slices defined by previous current edges. Directing the control point along similar slices is wasteful as it results in the control point repeatedly moving over the same section of surface. This is why a slice line is used to define the slices. The slice line changes only when the current edge is significantly different from the slice line.

During current edge following, the control point is made to stay close to the current edge. This is necessary to ensure that the control point moves over the unexposed surface near the current edge. To achieve this, current edge following checks the distance of the control point from the current point. Recall that the current point is the current edge point nearest the centre of the current slice. If the distance of the control point from the current point is greater than ρ then the behaviour of current edge following depends on the status of a flag. This flag is set whenever a new current slice is chosen, and cleared when the control point is within a distance ρ of the current point. If the distance of the control point from the current point is greater than ρ then the flag is examined:

- If the flag is set then the surface following direction is a vector from the control point towards the current point, as shown in Figure 34(a). Thus if the control point has not yet been within a distance ρ of this particular current point then the control point is directed towards the current point. Once the control point is within a distance ρ of the current point, the flag is cleared, and the control point is directed along the current slice.

- If the flag is cleared then the current slice is deemed used, and a new current slice is chosen thus setting the flag. The current edge following procedure then restarts. Thus if the control point has been within a distance ρ of this particular current point, and has since moved to a distance greater than ρ from the current point then the current slice is deemed used.

Once the control point is within a distance ρ of the current point, it is directed along the current slice. If the control point is in the current slice then the surface

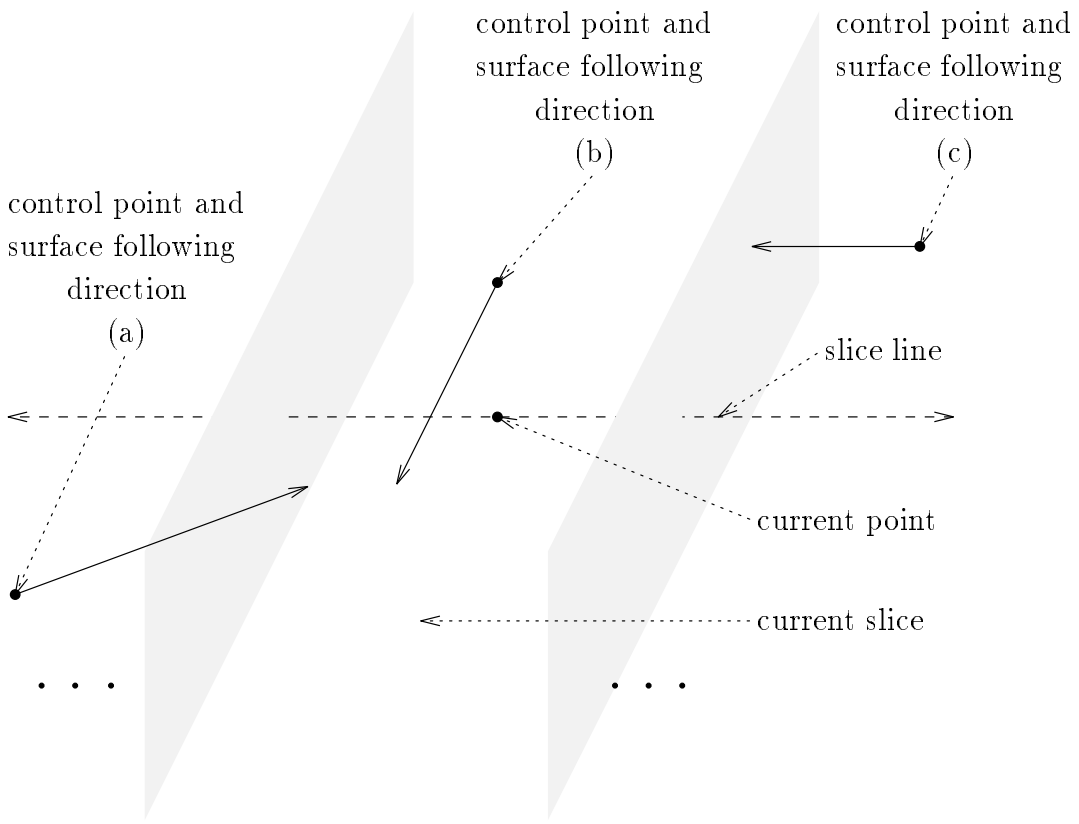


Figure 34: Current edge following. (a) The control point is directed towards the current point. (b) The control point is directed along the current slice. (c) The control point is directed towards the current slice.

following direction is the cross product of the slice vector and a vector from the slice line to the control point, as shown in Figure 34(b). If the control point is outside of the current slice then it must be directed towards the current slice. To achieve this, the surface following direction is parallel to the slice line, and towards the current slice, as shown in Figure 34(c). To prevent the control point from overshooting the current slice, the step size σ for the next control point motion is adjusted. As mentioned in Section 3.2, σ is the length of the control point motion at each step. If the control point's distance from the centre of the current slice is less than $\sigma_{\max}$ then the step size for the next control point motion may be at most the control point's distance from the centre of the current slice.

If, while the control point is being directed along the current slice, a surface model gap is detected between the current point and the closest surface model

point then the motion mode switches to gap following. Surface model gaps are defined in Section 3.4.5. If the process of directing the control point along the current slice, or the process of gap following does not result in surface modelling changing the current edge then the current slice is deemed used, and a new current slice is chosen.

3.4.4 Point Following

Point following is a special case of current edge following in which the current segment is a point segment rather than a planar polygon segment. It is often the case that when the current surface is chosen, it consists of a single point segment. This point segment becomes the current segment, so to complete the current segment it must first become a planar polygon segment. To achieve this the control point is directed over the unexposed surface that surrounds the current segment. This is intended to result in sensor points being detected near the current segment so that surface modelling can transform the current segment into a planar polygon segment.

Four slice lines passing through the current segment's description, and perpendicular to the line from the current segment's description to the control point, are defined. Their slice vectors form opposite and orthogonal pairs:

$$\mathbf{v}_1 = -\mathbf{v}_2,\ \mathbf{v}_3 = -\mathbf{v}_4,\ \mathbf{v}_1.\mathbf{v}_3 = 0,$$

where $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$ are the slice vectors. A single slice that has infinite width is defined for each slice line. An unused slice is chosen as the current slice, and the control point directed along it, as in current edge following. The surface following direction is the cross product of the slice vector and a vector from the slice line to the control point, as shown in Figure 34(b). The infinite width of the slices means that the control point is always within the current slice. Unlike current edge following there is no limit on the distance between the control point and the current point; the current point being the point segment.

If, while the control point is being directed along the current slice, a surface model gap is detected between the current point and the closest surface model

point then the motion mode switches to gap following. If motion of the control point along the current slice, or during gap following does not result in the current segment becoming a planar polygon segment then the current slice is considered used, and a new current slice is chosen from the remaining unused slices. If all four slices are used without the current segment becoming a planar polygon segment then the current segment is marked unclosable, and surface modelling and surface following stop. However, this has never occurred in any of the experiments that have been performed. Directing the control point along the slices has always resulted in the current segment becoming a planar polygon segment. Once the current segment becomes a planar polygon segment, the choice of current edge is cleared, and the motion mode switches to current segment following, where a new current edge is chosen. The slice line is not retained.

3.4.5 Gap Following

If, during current edge following or point following, a surface model gap is detected between the current point and the closest surface model point then the motion mode switches to gap following. Gap following directs the control point's sensors over the gap. This is intended to result in sensor points being detected in the gap so that surface modelling can close the gap. A surface model gap is deemed to lie between the current point and the closest surface model point if the following are satisfied:

- the control point is within the current slice, and
- the closest surface model point lies outside of the current edge (refer to Section 3.4.3).

Once a surface model gap is found, the motion mode switches to gap following, where the control point is directed over the gap. The gap is defined by the line segment between the current point and the *gap point*. The gap point is a point on the model segment closest to the control point. This closest model segment also contains the closest surface model point. To determine the gap point:

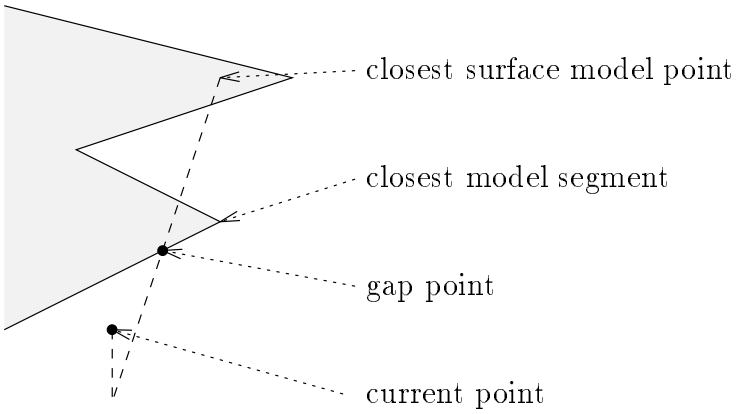


Figure 35: The gap point.

1. If the closest model segment is a point segment then the gap point is the closest surface model point.

2. If the closest model segment is a planar polygon segment then the current point is orthogonally projected onto the closest model segment's plane. A line segment is then formed between the current point's projection and the closest surface model point. If the line segment intersects the closest model segment's perimeter then the gap point is the intersection point closest to the current point's projection, as shown in Figure 35. If not then the gap point is the closest surface model point.

A simpler choice of gap point would be the closest surface point. However, the gap defined by this choice might be longer than necessary as it might pass over parts of the closest model segment, as in Figure 35, that is, the gap might include sections of the surface, so it is more like a series of gaps. Defining the gap point as described above ensures that this does not occur.

A series of *target points* is set along the line segment between the current point and the gap point, as shown in Figure 36. The target points start at the gap point, and are spaced apart a distance equal to half the surface model's linear resolution. Each target point is handled in turn starting with the target point that lies at the gap point, and working towards the current point. The control point is directed such that one of its sensors' sensing axes is moved towards the current target point.

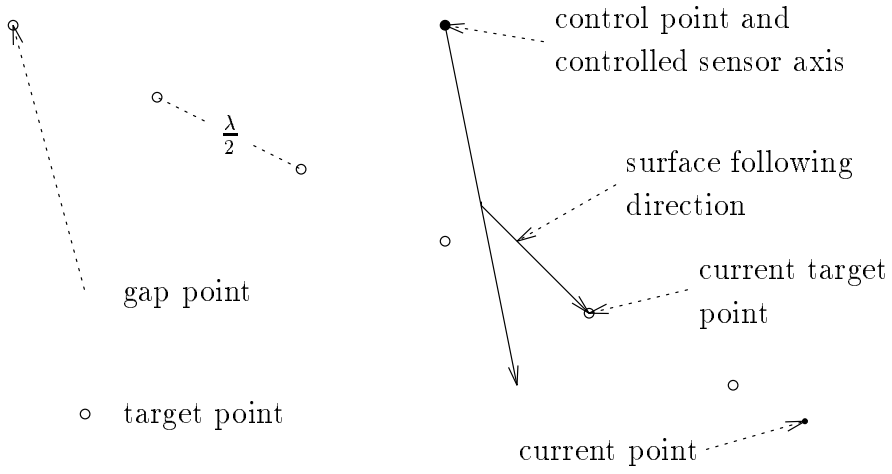


Figure 36: Aiming the controlled sensor at the target point.

This is intended to result in a sensor point being detected near the target point. If a sensor point is detected near the target point then gap following moves on to the next target point. A sensor point is considered to have been detected near a target point if it is a distance less than half the surface model's linear resolution from the target point. If, after some time, no sensor point has been detected near the target point then gap following gives up on the target point, and moves on to the next target point.

When gap following selects a target point, it chooses the sensor to be controlled. This sensor is called the *controlled sensor*. The controlled sensor is the sensor whose sensing axis is closest to the target point. This sensor remains the controlled sensor until gap following moves on to the next target point. The controlled sensor's sensing axis must be directed towards the target point. To achieve this, the sensing axis point closest to the target point is determined. The surface following direction is then the vector from the closest axis point towards the target point, as shown in Figure 36. The distance between the closest axis point and the target point is often quite small. To prevent the controlled sensor from overshooting the target point, the step size σ for the next control point motion is adjusted. If the closest axis point's distance d from the target point is less than $\sigma_{\max}$ then the step size for the next control point motion may be at most d.

If surface modelling changes the current edge then the gap has probably been closed, and so the motion mode switches to current segment following, where a new current edge is chosen. Similarly, if gap following was called from point following, and the current segment has become a planar polygon segment then the gap has probably been closed, and so the choice of current edge is cleared, and the motion mode switches to current segment following, where a new current edge is chosen. If all of the target points are used without surface modelling changing the current edge (or the current segment becoming a planar polygon segment) then the gap has probably not been closed, and so the current slice is deemed used. The choice of follow edge is cleared, and the motion mode switches to current segment following, where a new follow edge and current slice are chosen. Thus if the gap is closed then surface following moves on to a new current edge, if not then it persists with the current edge but moves on to a new slice passing through the current edge.

3.4.6 Surface Modelling Changes

During surface modelling, the model segments involved in successful mergers are recorded. This allows surface following to determine the changes that occur to the current segment. If the current segment is successfully merged with another model segment then the current segment becomes the model segment resulting from the merger. For example, if the current segment is merged with a model segment S_1 resulting in the formation of a new model segment S_2, and S_2 is subsequently merged with another model segment S_3 resulting in the formation of a model segment S_4 then the current segment becomes S_4.

Surface modelling also records the deletion of redundant model segments. This allows surface following to determine whether the current segment has been deleted. If so then the motion mode switches to current surface following, where a path to a new current segment is determined. If deletion of the current segment results in deletion of the current surface then surface modelling and surface following stop.

Surface modelling also records the edges changed by various surface modelling operations. This allows surface following to determine the changes that occur to edges that are of interest. The current edge, follow edge and path edges are edges

of interest. Edges are changed when surface modelling removes perimeter sections. Perimeter sections are removed in the following circumstances:

- When a perimeter section is extended to a projected point during the merger of a point segment and a planar polygon segment (refer to Section 2.4.2).
- When a perimeter section is extended to another perimeter section during the merger of two non-overlapping planar polygon segments (refer to Section 2.4.3).
- When the outermost perimeter is constructed, and holes' perimeters are collapsed during the merger of two overlapping planar polygon segments (refer to Section 2.4.3).
- When a perimeter section is extended to an intersection line to form a shared edge (refer to Section 2.6).
- When a fissure is filled (refer to Section 2.5).
- When a corner vertex is formed (refer to Section 2.7).
- When a redundant model segment is deleted (refer to Section 2.9).

Projecting a perimeter onto a plane of best fit, and repositioning shared edges do not constitute changes to edges. These operations merely reposition edges.

3.4.7 Surface Following Progress

Sometimes surface following does not make progress towards completion of the current surface. This can occur for the following reasons:

- No suitable reconfiguration vector exists for the robot. This usually occurs with fixed base robotic manipulator arms, for example, if the links of the robot are wrapped around a circular object. Determining a motion out of such configurations is not always simple and may require the control point to be moved away from the surface.

- In gap following the step size may be very small. This occurs if the controlled sensor's sensing axis is very close to the target point and no sensor points are detected near the target point.

- Surface following may direct the control point in a loop. For example consider the following situation: The motion mode is current segment following, and the current segment is not the model segment that is closest to the control point. At step i the surface following direction is towards and along the follow edge, as shown in Figure 37(a). Suppose that there is a large angle between the follow edge and the closest model segment, such that the resulting control point motion moves the control point away from the follow edge. Then at step $i+1$ the surface following direction will be more towards the follow edge than the previous surface following direction, as shown in Figure 37(b). Due to the large angle between the follow edge and the closest model segment, the resulting control point motion may then reverse the previous control point motion so that the control point is returned to the position it held at step i. This results in the control point looping between these two positions. Similar looping behaviour can occur for other surface following situations.

These situations occur infrequently but when they occur they must be detected and appropriate action taken.

A step counter is used to detect when surface following is making insufficient progress. Every τ steps the robot's configuration is stored. If the distance between the stored configuration and the previous stored configuration is less than a preset distance then surface following is deemed to have made insufficient progress. The preset distance used is the ceiling value γ placed on the length of computed actuator changes (refer to Section 3.3). If the robot has both prismatic and revolute joints then the stored configurations are partitioned into their prismatic and revolute components, and the preset distances are the ceiling values γ_{p} and γ_{r}, respectively.

The robot often changes the direction of its motion when surface following changes the part of the surface model to be followed. This can reduce the distance between successive stored configurations, and in the worst case could result

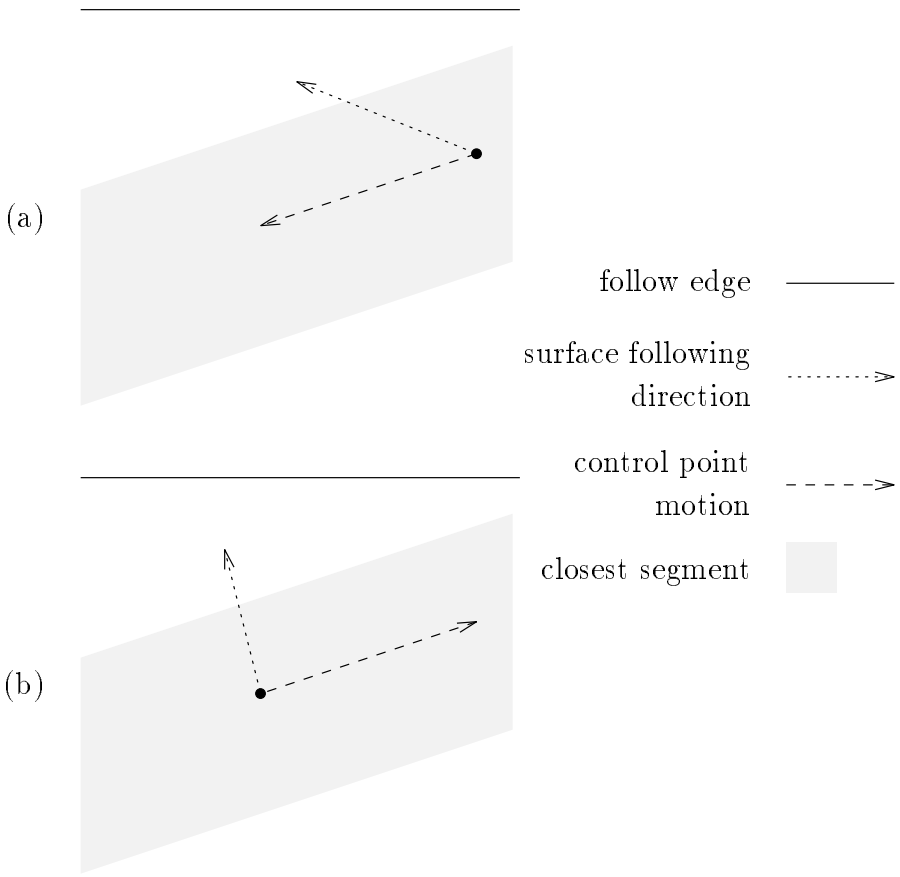


Figure 37: Looping behaviour. (a) The control point moves along the closest model segment. (b) The control point is returned to it previous position.

in surface following being deemed to be making insufficient progress. To avoid this, on the following occasions the step counter is reset to zero, and the robot's configuration is stored:

- In current surface following and current segment following whenever a new follow edge is chosen.
- In current edge following and point following whenever a new current slice is chosen.
- In gap following whenever a new target point is chosen.

Note that these situations correspond to surface following making progress towards completion of the current surface.

If surface following is making insufficient progress then appropriate action must be taken. The action taken amounts to surface following giving up on its current goal and choosing a new goal:

- If current surface following is making insufficient progress then the control point cannot follow the path to the current segment any further, and so the current segment is marked unreachable, and current surface following restarts. This results in a new path being planned to a new current segment.

- If current segment following is making insufficient progress then the control point cannot follow the current segment's perimeter to the current edge any further, and so the current edge is marked unreachable, and current segment following restarts. This results in a new current edge being chosen.

- If current edge following or point following are making insufficient progress then the control point cannot move along the current slice any further, and so the current slice is deemed used, and a new current slice is chosen.

- If gap following is making insufficient progress then the controlled sensor's sensing axis cannot be brought any nearer the target point, and so a new target point is chosen.

Note that these actions all result in surface following choosing a new goal, and so progress towards the completion of the current surface is made. However, the new goal is not guaranteed to ensure progress of the robot's control point along the surface. For example, the robot may be stuck in a configuration for which no suitable reconfiguration vector exists without moving the control point away from the surface. Thus no surface following motion will move the robot out of this configuration, and so no choice of new goal will help. In such cases new goals will be chosen and subsequently marked unreachable until no incomplete model segments remain, that is, the surface model is complete. This is how surface following guarantees completion of the surface model and thus termination of the algorithm. However, this does not necessarily imply complete coverage of the surface.

In most cases, completion of the current surface results in the modelling of as much of the surface as possible. However, the completed current surface may have

unreachable model segments and edges, and these may represent unmodelled parts of the surface. Furthermore, the robot may be able to move its control point to these model segments and edges. Nevertheless, it is guaranteed that if the current surface is complete then surface following has at least attempted to move the control point to such model segments via a shortest edge-path, and to such edges via model segment perimeters. It is recommended that upon completion of the surface model, a sensor-based motion planner be employed to reposition the control point near any unreachable model segments or edges. Such a path planner need not be constrained to following the surface. Once the control point is repositioned near such a model segment or edge then surface modelling and surface following can recommence. This issue is not examined in this thesis.

The completed current surface may also have unclosable edges that surface modelling may be able to change if more sensor points are obtained. Nevertheless, it is guaranteed that if there is a substantial area of unexposed surface near such edges then surface following has at least attempted to obtain sensor points in the proximity of them.

3.5 Summary

Each control point motion aims to maintain a desired distance between the robot's control point and the surface model, and the length of each control point motion equals a set step size. These measures ensure that the control point's sensors remain within sensing range of the surface, and that the control point is prevented from colliding with the surface.

Redundancy is used to prevent the rest of the robot from colliding with the surface. Homogeneous components of a reconfiguration vector are used to keep the robot's links away from the surface model. The reconfiguration vector is obtained from the gradient of an objective function that is based on the distance between the robot's links and the surface model.

A surface following direction forms a component of each control point motion. The surface following directions direct the robot's control point to non-shared edges,

and once at a non-shared edge, direct the control point over the unexposed surface near the edge. The surface following directions are computed by various motion modes:

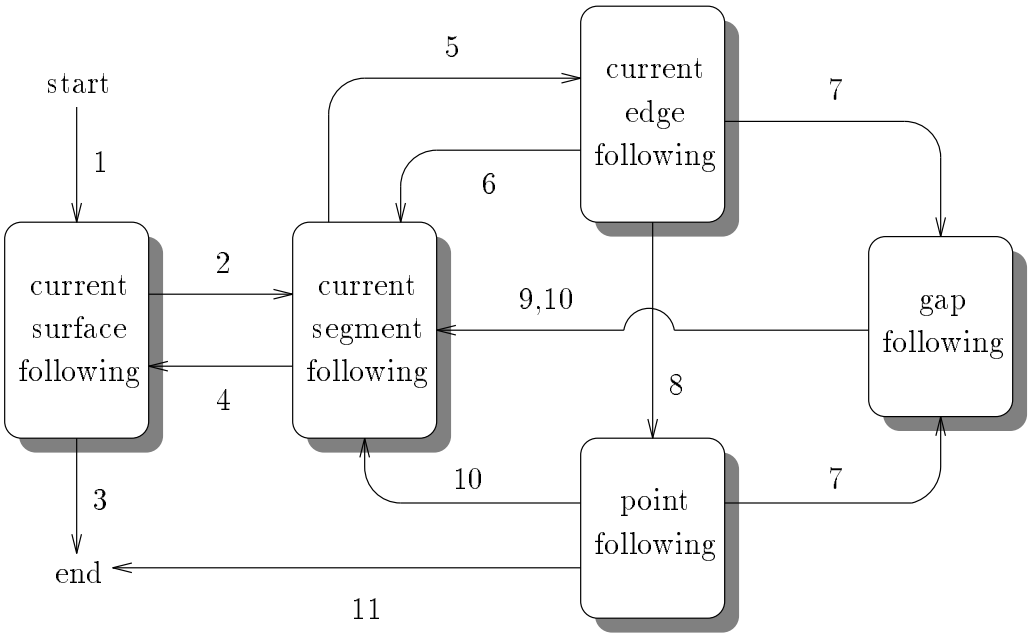
- Once an incomplete surface has been selected, current surface following directs the control point to each of the surface's incomplete model segments. Surface following directions are computed that direct the control point to a particular model segment via the shortest edge path.

- Once at an incomplete model segment, current segment following directs the control point to each of the model segment's incomplete edges. Surface following directions are computed that direct the control point to a particular edge via the model segment's perimeter.

- Once at an incomplete edge, current edge following directs the control point over the unexposed surface near the edge. The unexposed surface is divided into slices, and surface following directions are computed that direct the control point along each slice. Slices that contain very little unexposed surface are ignored. Slices are retained between similar non-shared edges to avoid repetitive motion of the control point along similar slices. The special case of the model segment being a point segment is also handled.

- If while the control point is being directed along a slice, a surface model gap is detected near the control point then gap following directs the control point's sensors over the gap. The gap is divided into target points, and surface following directions are computed that direct the control point's sensors to each target point.

Figure 38 shows the switching between motion modes. If surface following is making insufficient progress towards the completion of the surface then the current goal is abandoned by marking it unreachable or unclosable, and a new goal is chosen. This effectively forces progress to be made, and guarantees the completion of the surface. Completion of the surface usually results in but does not guarantee complete coverage of the surface by the robot's sensors.



1. Surface following starts.
2. The control point reaches the current segment.
3. The current surface is complete or deleted.
4. The current segment is complete.
5. The control point reaches the current edge, or the follow edge is incomplete.
6. The choice of current edge cleared, or the current edge is complete.
7. A gap is detected near the control point.
8. The current segment is a point segment.
9. The choice of current edge is cleared, or all of the target points have been used.
10. The current segment is no longer a point segment.
11. All of the slices have been used.

Figure 38: Motion mode switches. Note that the numbers on the arcs are merely labels, they do not represent an ordering of the mode switches.

Chapter 4

Results and Discussion

4.1 Introduction

To evaluate the surface modelling and surface following techniques presented in Chapters 2 and 3, a robot equipped with range sensors is required. This chapter describes the implementation of range sensors, and presents and discusses some surface modelling and surface following results that use real sensor data.

The layout of this chapter is as follows: The implementation of range sensors is described in Section 4.2. Section 4.3 presents and discusses the results of surface modelling and surface following for a variety of robots and surfaces using real sensor data, and the chapter concludes in Section 4.4.

4.2 Single-Point Range Finders

The surface modelling and surface following techniques presented in Chapters 2 and 3 are for use with robots equipped with single-point range finders. Single-point range finders can be realised with lasers. Unfortunately, the cost of laser range finders makes mounting a number of them on a robot prohibitively expensive. Most laser range finders determine the distance to a surface by measuring the phase shift between the light emitted by the laser and the light reflected by the surface. A cheaper alternative that does not require the use of lasers is to determine the distance by measuring the intensity of the reflected light.

Experiments were performed using pairs of spectrally matched infra-red emitters and detectors. The intensity of the reflected infra-red light detected by the detector was used to measure the distance to the reflecting surface. It was found that the accuracy of this measurement varied greatly with the angle of incidence of the emitted light at the reflecting surface. To reduce this effect, the beam of emitted light was narrowed by hooding the emitter. However, this reduced the intensity of the reflected light to almost undetectable levels. Even when high intensity, narrow beam-angle, hooded emitters were used, the effect of the angle of incidence on the accuracy of the distance measurement was significant.

An optical displacement sensor from Hamamatsu was also investigated. This sensor consists of an infra-red emitter and a segmented photodiode array. The distance to the reflecting surface is determined from the position along the array of the highest intensity reflected light. However, it was found that the accuracy of this device also depends on the angle of incidence of the emitted light at the reflecting surface.

Rather than trying to develop a single-point range finder, when perfectly adequate though expensive devices already exist in the form of laser range finders, it was decided that range images produced by laser range scanners would be used to provide sensor data for evaluating surface modelling and surface following. The range images are sets of points (x, y, z). By placing a simulated robot in the same coordinate frame as the range image, points from the image can be selected as sensor points for use with surface modelling. Given the robot's configuration, the location of its range sensors' sensing axes can be determined. For each sensing axis, the range image points within a small distance ϵ of the axis are determined. Of these, the range image point nearest the sensor end of the sensing axis becomes a sensor point. The distance

$$\epsilon = \max\{\min\{\|\mathbf{r}_i - \mathbf{r}_j\|,\ j = 1, \ldots, n,\ i \neq j\},\ i = 1, \ldots, n\},$$

where $\mathbf{r}_i$, for $i = 1, \ldots, n$ are the range image points.

There is noise in laser scanner images that would be similar to the noise of single-point laser range finders. Other sources of noise and error in the measurements of robot-mounted range sensors would be due to noise and errors in the robot's joint

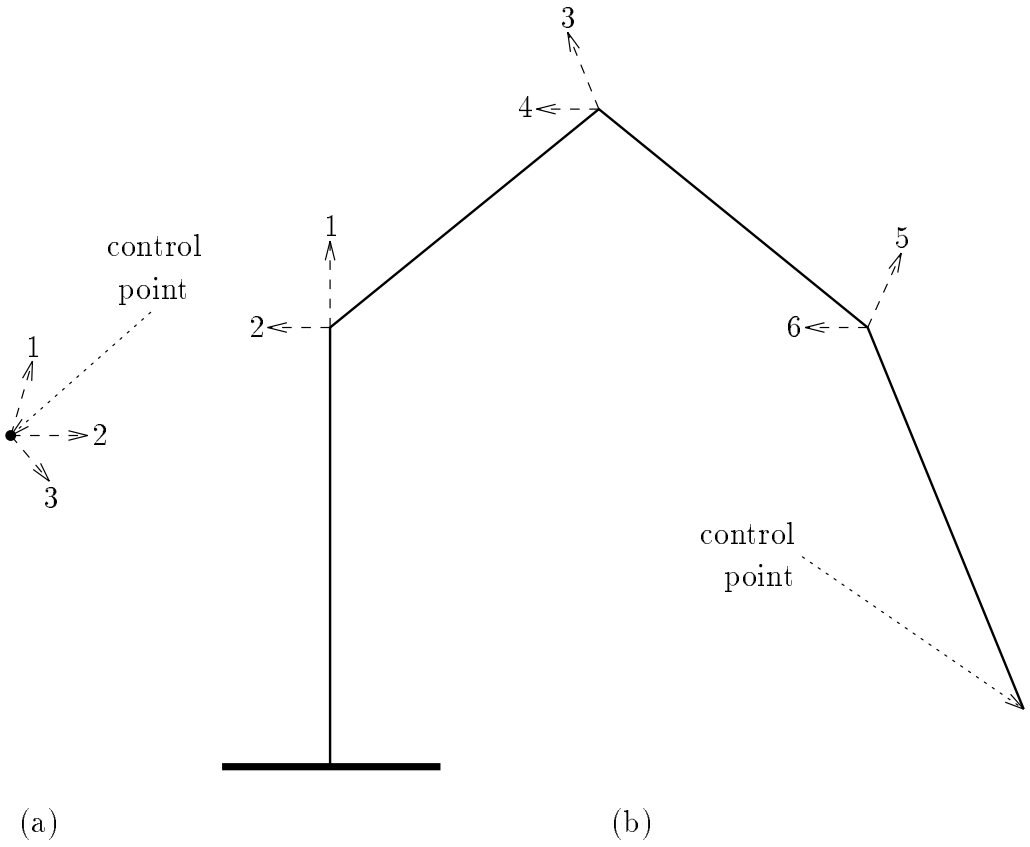


Figure 39: The robots: (a) A mobile point robot. (b) A four-link manipulator arm. The dashed lines indicate the joint axes.

encoders. These types of error are not represented in the results that follow and should be the subject of further research.

4.3 Results and Discussion

Results are presented for two simulated robots: a mobile point robot with three orthogonal, prismatic degrees-of-freedom, as shown in Figure 39(a), and a four-link manipulator arm with six revolute degrees-of-freedom, as shown in Figure 39(b). Although most mobile robots are restricted to two-dimensional motion, mobile robots with three degrees-of-freedom do exist, for example, submersible robots. Also mobile robots for space exploration will require three degrees-of-freedom. The manipulator arm's links are of length 1. The robots' control points are the mobile robot's centre, and the manipulator arm's end-effector. Both robots can move their control

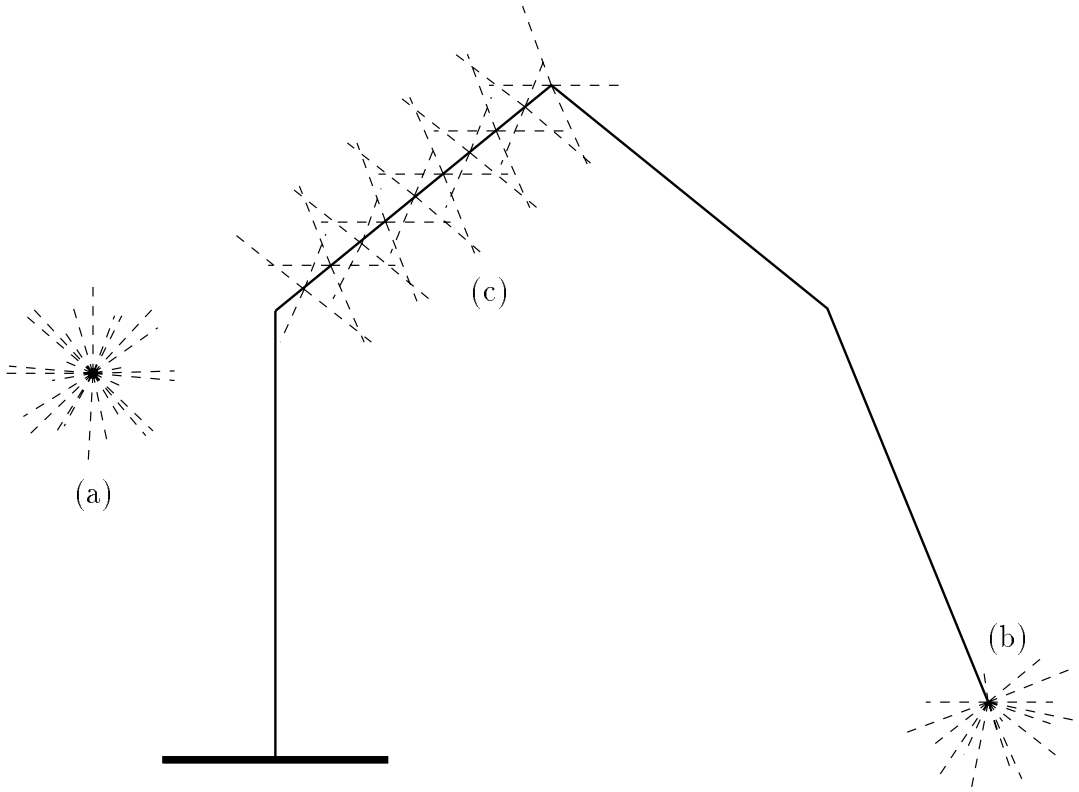


Figure 40: The range sensors of the (a) mobile robot, (b) the manipulator arm's control point and (c) second link.

points in three-dimensions. Range sensors are mounted around the robots' control points, as shown in Figure 40. The mobile robot and manipulator arm have 26 and 17 range sensors, respectively, mounted around their control points. An additional 116 range sensors are mounted on the manipulator arm's second (40), third (40) and fourth (36) links. As the manipulator arm's first joint is at the end of its first link, the first link is immobile, and so requires no range sensors. Figure 40(c) shows the second link's range sensors. Similar sensor arrangements are mounted on the third and fourth links. The range sensors of both robots have a sensing range of 0.2.

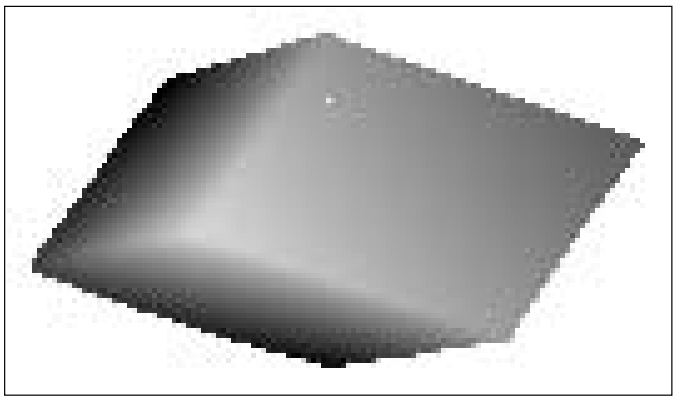
To perform surface modelling and surface following, each robot was placed in the same coordinate frame as a range image, with the robot's control point within sensing range of some range image points. The range images were produced by the Michigan State University Pattern Recognition and Image Processing Laboratory's

Technical Arts 100X scanner. Copies of the images were obtained from a public repository maintained by Michigan State University[1]. The scanner is a fixed sensor, and so suffers from occlusion. To provide surface modelling and surface following with solid objects, for each range image, the occluded surfaces were estimated by manually identifying the range image points that represent object vertices and lie on the occlusion boundary. These vertices were used to form a set of planar polygonal model segments called *initial* model segments. Thus the initial model segments estimate the occluded surface of the objects in the range images. These initial model segments are like ordinary model segments except that they represent no sensor points, and so their model segment data are zero. As merger operations make use of model segment data, initial model segments cannot be merged with other model segments, and so cannot be marked as having been modified. However, shared edges and corner vertices can be formed in the perimeters of initial model segments.

Figures 41, 42 and 43 show the range images and initial model segments of three objects named Cube, Block1 and Block2, respectively. The range images' ϵ values are also given. Note that the range images show only the z coordinate, so their perspective is incorrect. Also the background has been removed from each image.

Figure 44 shows the evolution of the surface model constructed as the mobile robot moves over the surface of Block1. The number of sets of sensor points processed so far is given at each stage of the evolution. Note that the initial model segments are not shown. Figure 45 shows the path of the mobile robot's control point during the construction of the surface model of Block1. The path is superimposed on the completed surface model of Block1. Figure 46 shows the evolution of the surface model constructed as the manipulator arm moves over the surface of Block1. Figure 47 shows the path of the manipulator arm's control point during the construction of the surface model of Block1. The path is superimposed on the completed surface model of Block1. Figures 48(a) and (b) show the completed surface models of Cube and Block2, respectively, for the mobile robot, and Figures 49(a) and (b) show the completed surface models of Cube and Block2, respectively, for

[1]These images are available via anonymous ftp from irl.eecs.wsu.edu:/pub/range-images.



$\epsilon = 0.0239$

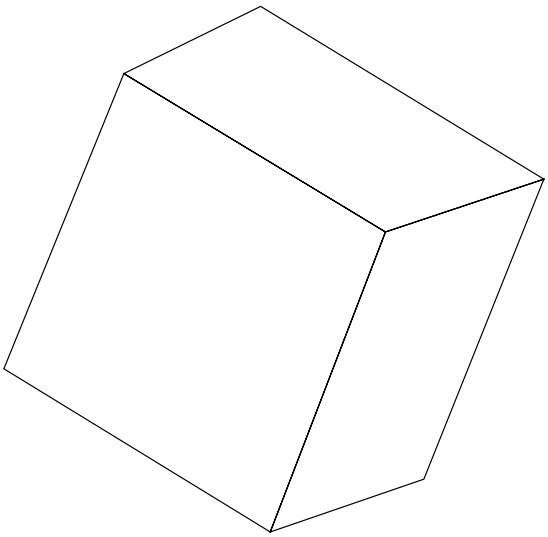
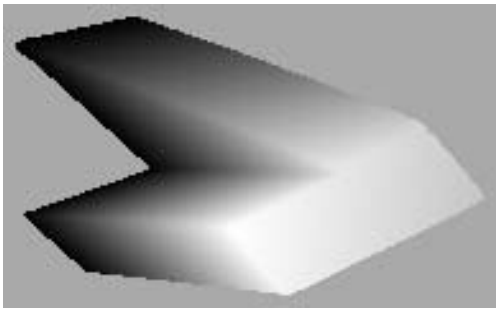


Figure 41: The range image and initial model segments of Cube.



$\epsilon = 0.0118$

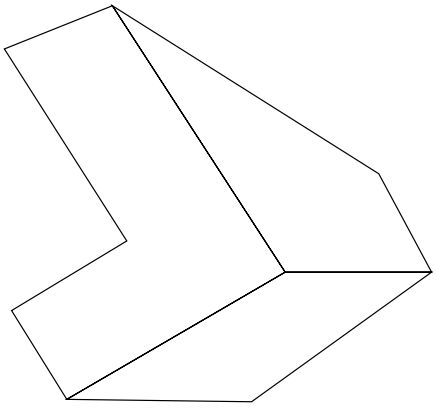


Figure 42: The range image and initial model segments of Block1.

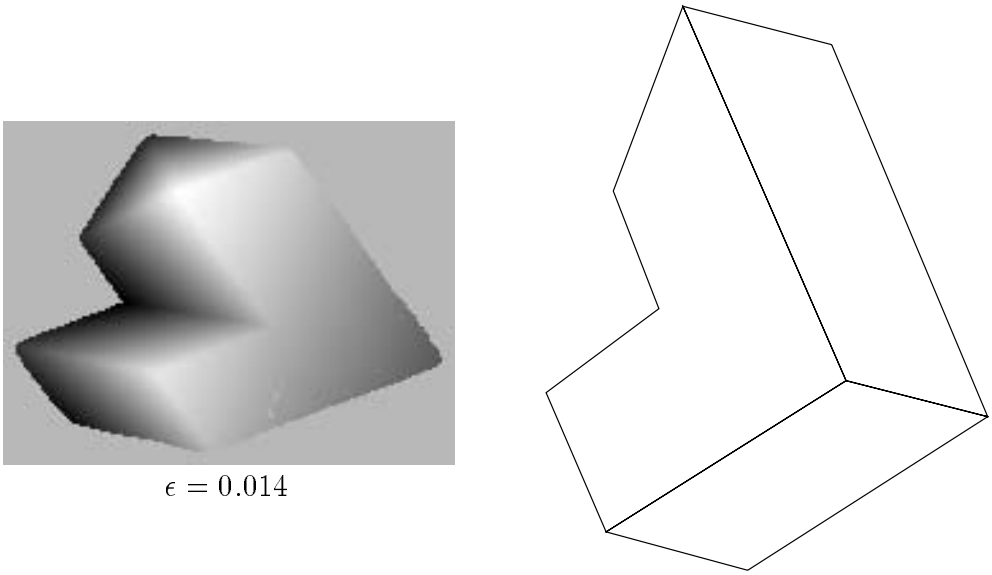


Figure 43: The range image and initial model segments of Block2.

the manipulator arm.

The robot paths in Figures 45 and 47 are different due to the robots' differing kinematics and sensor arrangements. Although the paths to the current segments are via shortest edge-paths and the paths to the current edges are via the current segment's shortest perimeter section, the overall paths are by no means optimal. To construct an optimal path, full *a priori* knowledge of the surface is required. This is not the case for surface modelling problems.

The measurements and positions of the objects in the range images are not known, so precise measurements of the surface models' accuracy cannot be determined. However, it can be seen that the completed surface models are good approximations of the surfaces present in the range images. In some cases there are redundant model segments that have gone undetected. Also there are some gaps that have not been filled either because shared edges could not be formed, or sensor points were not detected in the gaps. However, these deficiencies are only minor, and the completed surface models could be used for most subsequent robotic tasks.

The surface models were constructed with a linear resolution $\lambda = 0.05$. If λ is decreased then a more accurate surface model usually results. However, more sensor data is required to form the surface model because model segments must be closer

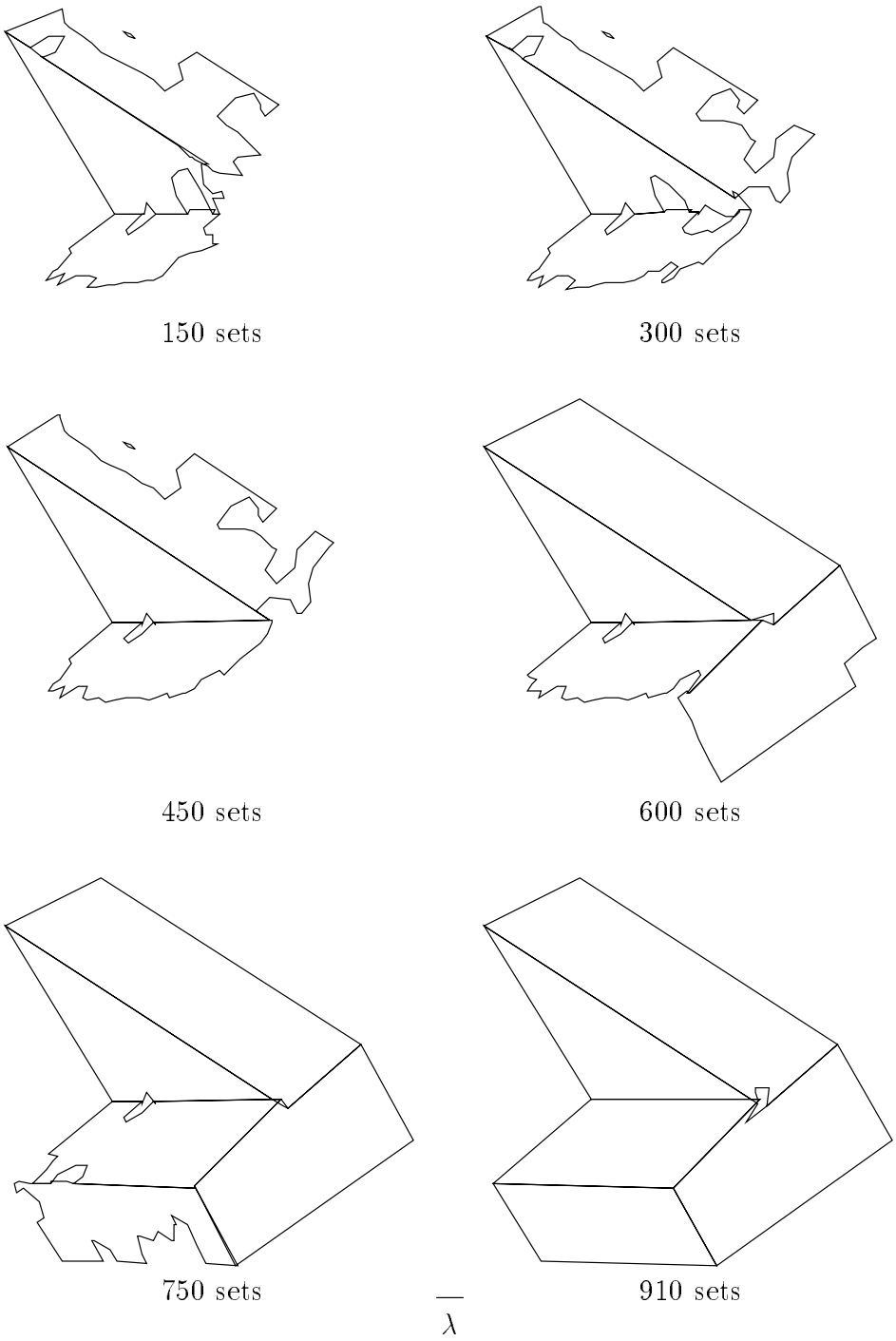


Figure 44: The evolution of the surface model of Block1 for the mobile robot.

together for mergers to be performed and for shared edges and corner vertices to be formed. The linear resolution also affects the choice of other parameters such as the surface following distance d. The angular resolution $\mu = 10°$. If μ is too large then

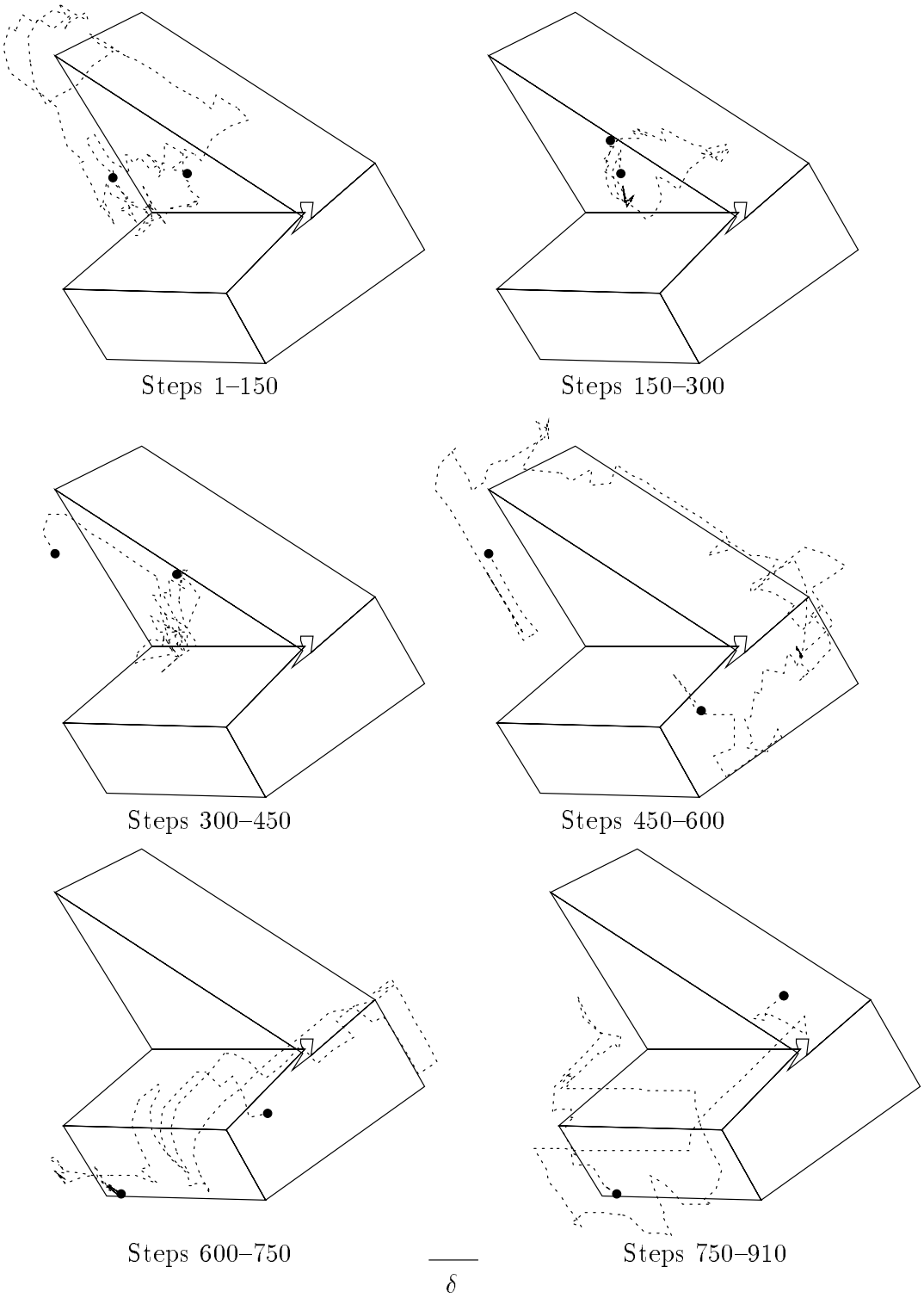


Figure 45: The path (dotted line) of the mobile robot's control point during the construction of the surface model of Block1.

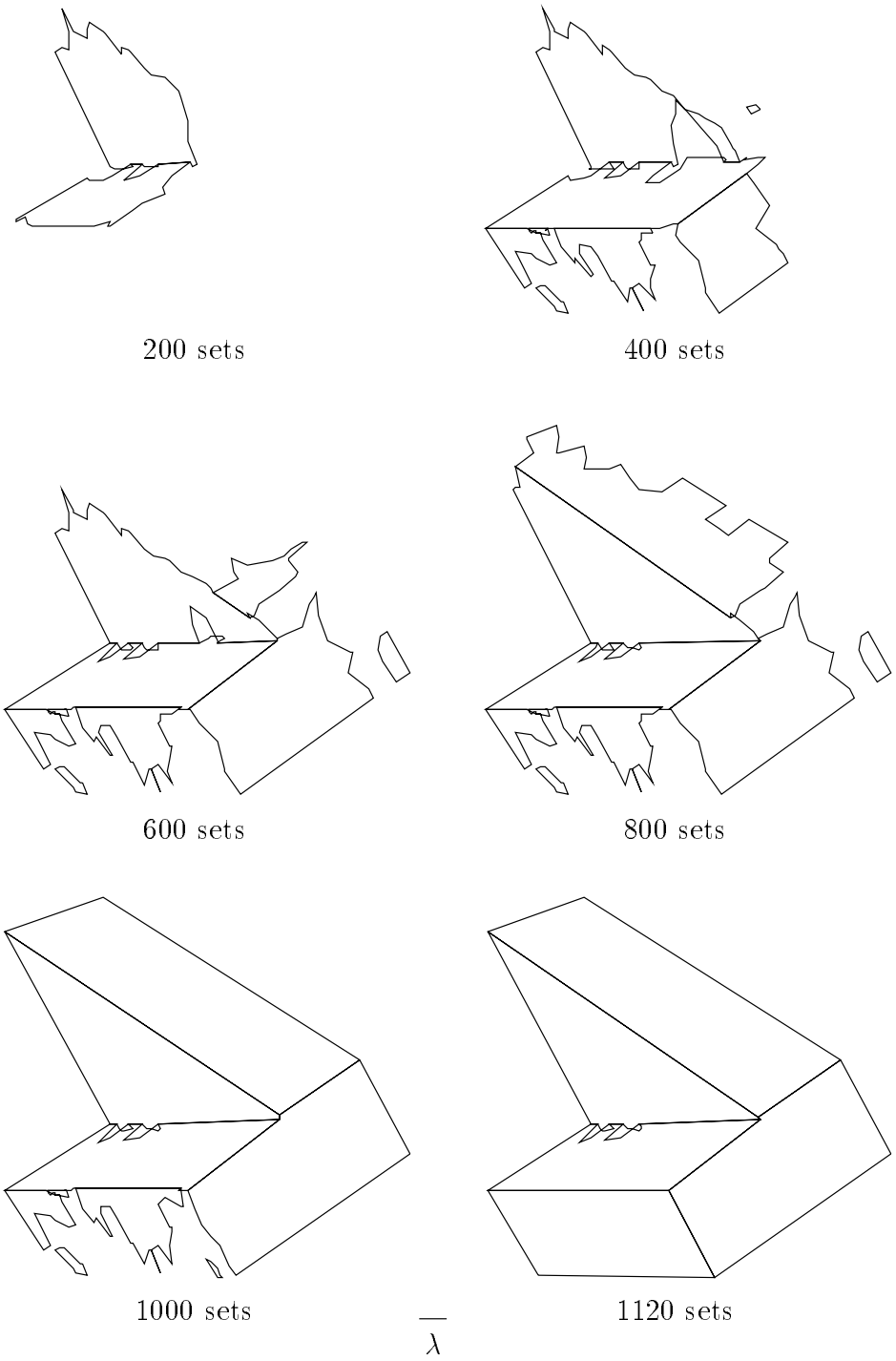


Figure 46: The evolution of the surface model of Block1 for the manipulator arm.

planar polygon segments that meet at large angles may be merged thus degrading the accuracy of the surface model. If μ is too small then few planar polygon segments are merged and the surface model accuracy suffers. The threshold values used to

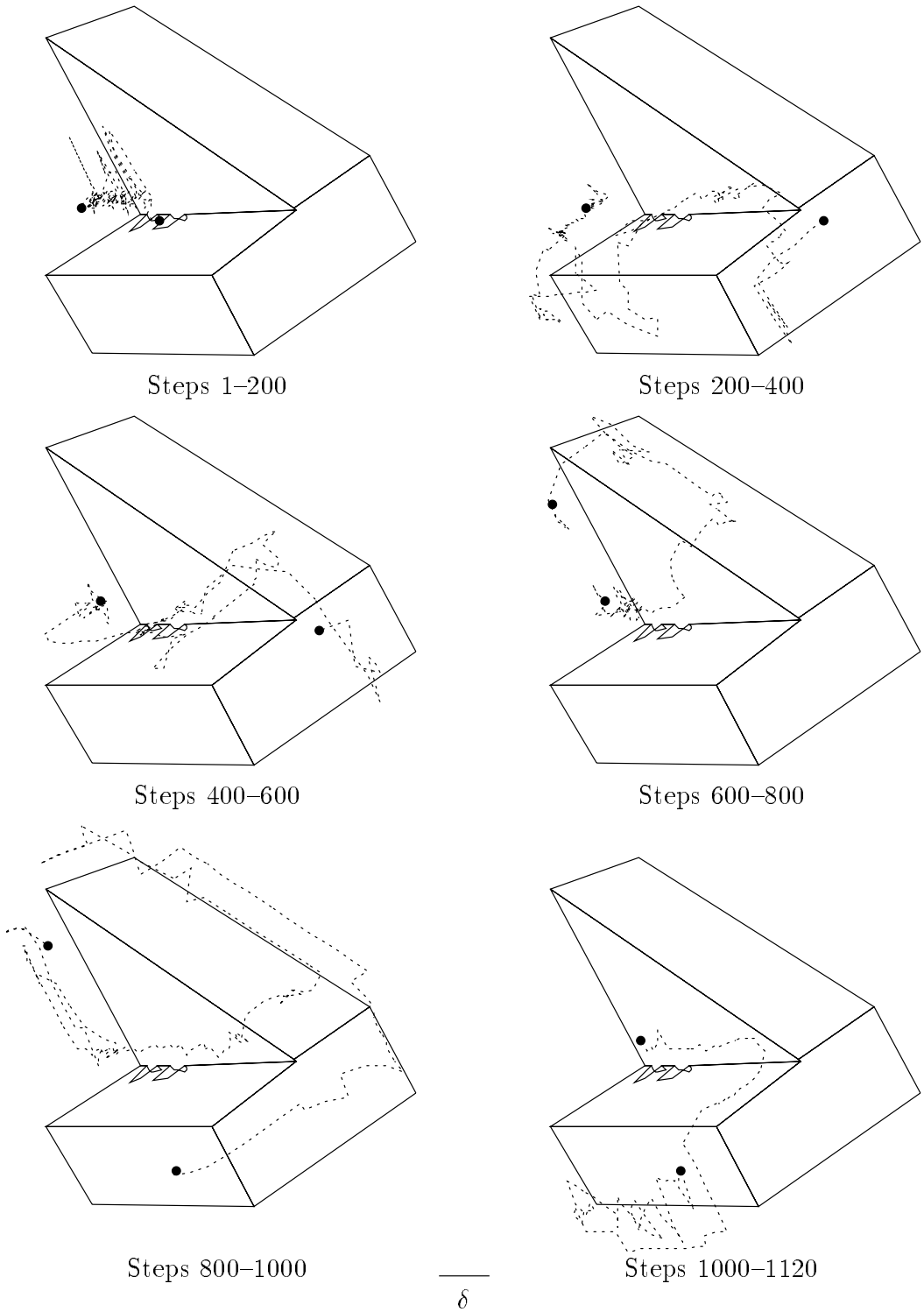


Figure 47: The path (dotted line) of the manipulator arm's control point during the construction of the surface model of Block1.

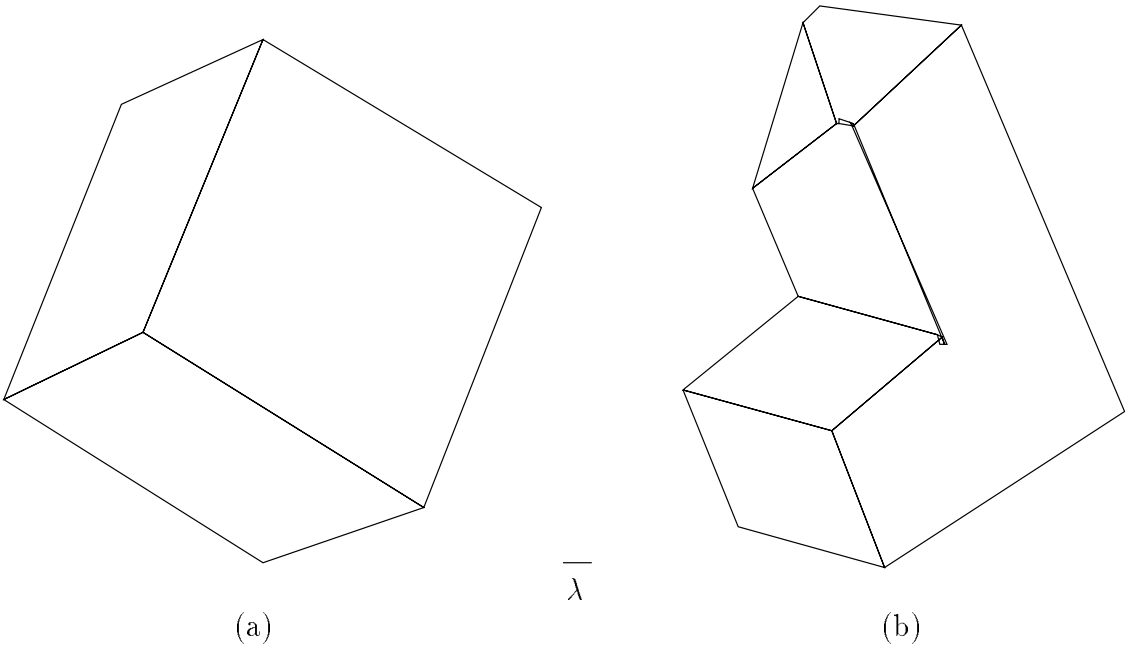


Figure 48: The completed surface models of (a) Cube and (b) Block2 for the mobile robot.

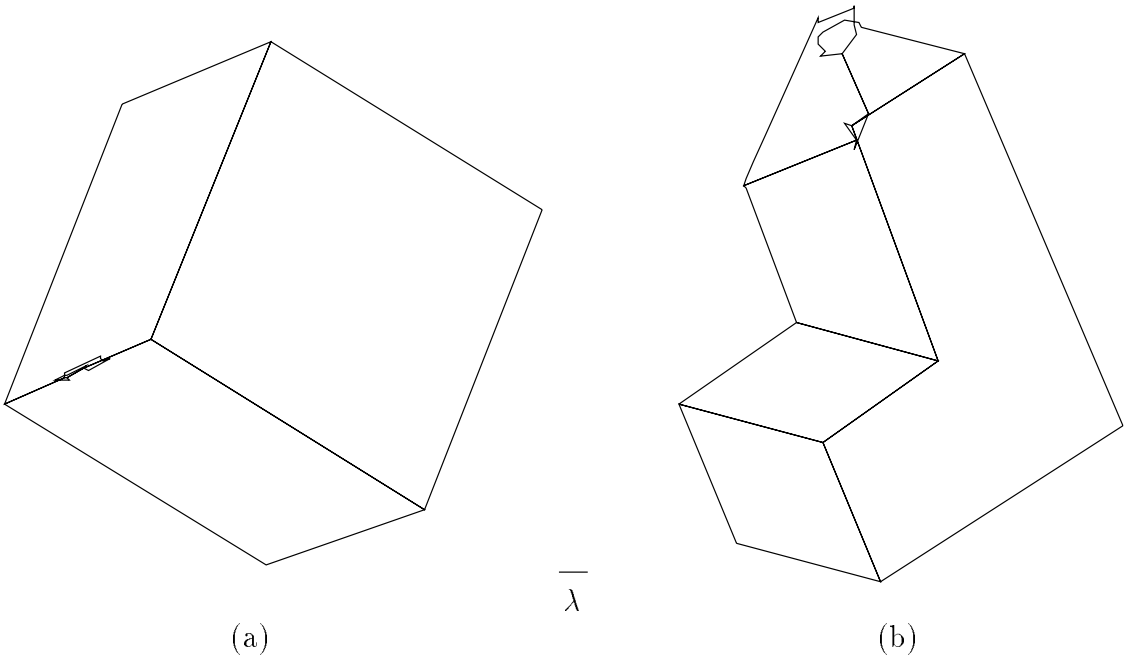


Figure 49: The completed surface models of (a) Cube and (b) Block2 for the manipulator arm.

detect redundant model segments were $\psi = 20$ and $\phi = 0.05$. The effects of varying ψ and ϕ were discussed in Section 2.9.

The desired distance to be maintained between the control point and the surface model was $\delta = 0.1$, less than the sensing range of the robots' range sensors, and greater than λ. This keeps the control point within sensing range of the surface and prevents the control point from colliding with the surface. Within this range, if λ is decreased then the danger of the control point colliding with the surface is increased. If λ is too large then operations such as current edge following and gap closing are adversely affected. These operations rely on motion of the control point resulting in similar motion of the sensor axes over the surface. If the control point is far from the surface then this relationship is unlikely to hold.

The maximum step size for the control point $\sigma_{\max} = 0.1$, less than the sensing range of the robots' range sensors. This ensures that the control point does not move out of the region sensed by the control point's sensors. Also $\sigma_{\max}$ was chosen such that successive sets of sensor points overlap. If $\sigma_{\max}$ is decreased then each motion is shorter and so more surface following steps are required to complete the surface. If $\sigma_{\max}$ is increased then the further apart are consecutive sets of sensor points resulting in more fragmented surface modelling.

The ceilings on the length of the vector of computed actuator changes for the mobile robot and the manipulator arm were $\gamma = 0.05$ and $\gamma = 3.8°$, respectively. These values of γ ensure that no actuator changes satisfying $\|\dot{\Theta}\| < \gamma$ will move any part of either of the robots a distance greater than the sensing range of the robots' range sensors. For the manipulator arm, γ was determined using the method described in Section 3.3. If γ is decreased then each robot motion is shorter and so more surface following steps are required to complete the surface.

The mobile robot is small enough such that maintaining δ between the control point and the surface model prevents the robot from colliding with the surface model. This is not the case for the manipulator arm. The arm's redundant degrees-of-freedom were used to prevent the arm from colliding with the surface model. A gain value of $\alpha = 7000$ was used in the computation of the arm's actuator changes (3), the exponent in the objective function (4) was $g = 6$, and the distance from

the surface model beyond which a link has no influence on the objective function was $\sqrt{2d_0} = 0.2$, the sensing range of the robots' range sensors, that is, a link had no influence on the reconfiguration of the arm if it was out of sensing range of the surface model. Neither robot collided with the surface model during the surface following examples presented in this chapter. The values of α and g were determined experimentally. If α is too small then not enough reconfiguration of the robot is included in the robot's motion and the danger of collision is increased. If α is too large then reconfiguration dominates the robot's motion. Varying g has similar effects on the robot's motion. If d_0 is decreased then the robot's links must be closer to the surface before they are reconfigured away from the surface. Thus the danger of collision is increased. If d_0 is too large then robot links that are far from the surface have an effect on the robot's motion.

The greatest distance that the control point was allowed from the part of the surface model being followed was $\rho = 0.2$, the sensing range of the robots' range sensors, that is, surface following attempted to keep the control point within sensing range of the part of the surface model being followed. If ρ is too small then the control point is made to follow the surface so closely that motion is restricted. If ρ is too large then the control point can wander far from the part of the surface model being followed, and so the surface following goal might not be reached.

To monitor the robots' surface following progress, their configurations were checked every $\tau = 10$ steps. This lead to the termination of the surface following examples presented in this chapter, that is, the surface models were completed. If τ is too small then surface following may be deemed to be making insufficient progress too frequently thus abandoning surface following goals too often. If τ is too large then it may take too long to recognise that insufficient progress is being made resulting in surface modelling and surface following taking a long time to complete.

The results presented in this chapter were produced by an implementation of surface modelling and surface following running on a Sun Sparcstation-2. The implementation was programmed using the 'C' programming language [56], and compiles to 283K of object code.

Table 1: Surface modelling and surface following performance.

	mobile robot		manipulator arm	
object	*sets*	*time (s)*	*sets*	*time (s)*
Cube	693	0.0753	779	0.2343
Block1	910	0.0779	1120	0.2833
Block2	938	0.1143	1422	0.1983

Table 1 summarises the performance of the surface following and surface modelling examples presented in this chapter. The *sets* columns give the number of sets of sensor points used to construct each surface model. Although the Cube is slightly larger than the Blocks, fewer sets of sensor points were used to construct its surface models than were used to construct the Blocks' surface models. This is because the Cube has fewer edges than the Blocks. For each edge on a surface, corresponding shared edges must be formed in the surface model. To form these shared edges, sensor points must be detected in gaps between model segments. To detect these sensor points, the gap following motion mode is used. Thus less gap following needed to be performed for the Cube than needed to be performed for the Blocks, and so fewer surface following steps were performed for the Cube than were performed for Blocks, resulting in fewer sets of sensor points being obtained for the Cube than were obtained for the Blocks. Also the number of sets of sensor points used to construct the surface models is greater for the manipulator arm than for the mobile robot. This is because some of the surface following motion of the manipulator arm was reconfiguration to avoid collisions with the surface model, whereas no reconfiguration was performed by the mobile robot.

The *time* columns give the average time taken to perform each surface modelling and surface following step, that is, the average time between receiving a set of sensor points, and the robot performing the next surface following motion. The average time is the total running time divided by the number of sets of sensor points used to construct the surface model. The total running time does not include the time spent selecting the sensor points from the range images. In general, the average time increases as the surface model complexity increases. Also the average times

are greater for the manipulator arm than for the mobile robot. This is because the manipulator arm's description is more complex than that of the mobile robot. The average times are quite low. If surface modelling and surface following were implemented for a real robot and range sensor system, most time would be spent moving the robot, and obtaining readings from the range sensors.

The use of initial model segments demonstrates the ability of surface modelling to use *a priori* information about the surface being modelled. When *a priori* surface information can be expressed in the form of (initial) model segments, surface modelling can fuse this information with sensor data obtained during surface following.

Results of surface modelling and surface following for other surfaces have been published previously [85,86,87]. These papers are included in Appendices C, D and E, respectively. These results were obtained using simulated range sensors and surfaces rather than range images. Surface modelling and surface following results for planar robots and surfaces have also been published previously [82,84,83].

The surface modelling and surface following examples presented so far in this chapter have involved the surfaces of planar polyhedra. As anticipated in Chapter 2, when dealing with curved surfaces, the surface model's accuracy is poor. Figure 50 shows the range image and initial model segment of an object with a curved surface, and Figure 51 shows the completed surface model of the object for the mobile robot. A total of 13004 sets of sensor points were used to construct the surface model. This is far greater than the number of sets of sensor points used to construct the surface models of Cube, Block1 and Block2. This is because many gaps had to be filled in the surface model of the curved surface. This resulted in much gap following being performed, during which many sets of sensor points were obtained. The average time to perform each surface modelling and surface following step was 0.5834 seconds. This is also greater than the average times for the Cube, Block1 and Block2. This is because of the very high complexity (in terms of the number of planar polygons, edges and vertices) of the surface model of the curved surface. Clearly, surfaces of a higher order than points and planar polygons are required to accurately model curved surfaces.

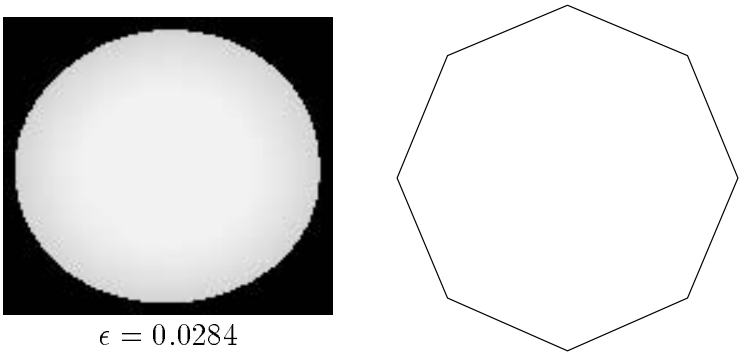


Figure 50: The range image and initial model segment of a curved surface.

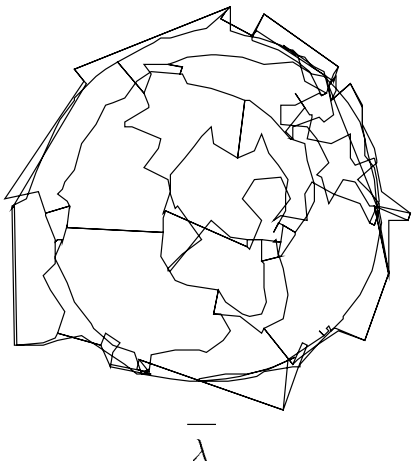


Figure 51: The completed surface model of a curved surface.

4.4 Conclusion

Results of surface modelling and surface following for a variety of robots and surfaces using real sensor data have been presented. These results demonstrate the effectiveness, when dealing with planar polyhedra, of the surface modelling and surface following techniques presented in this thesis. To accurately represent curved surfaces, model segments with high-order surface descriptions are required.

Chapter 5

Conclusion

5.1 Review and Contributions

The aim of the research presented in this thesis has been to automate the process of surface modelling, where the surface model is constructed from sensory information obtained from sensors moved over a surface. In Chapter 1 the mechanics of this automation were described. Range sensors are mounted on a robot that moves the sensors over the surface. The range sensors are single-point range finders that provide sets of sensor points, that is, the surface points they detect. The sets of sensor points obtained during the robot's surface following motion are used to construct a surface model. The surface model is used in turn in the computation of the robot's surface following motion. Thus surface modelling is performed on-line, that is, the surface model is constructed incrementally from the sensor points as they are obtained.

Chapter 2 introduced a surface model for use with incremental surface modelling. The surface model consists of a set of model segments, where each model segment has either a point or planar polygon description that is in some sense a "best fit" to the sensor points represented by the model segment. A set of sensor points is used to update the surface model by creating a point segment for each of the sensor points. The creation of these point segments may result in the formation of gaps in the surface model, so various model segment operations are performed in an attempt to fill these gaps, and reduce the complexity of the surface model:

- Two model segments may be merged to form a new model segment.
- Fissures in a planar polygon segment's perimeter may be filled.
- Shared edges may be formed between two planar polygon segments thus making them neighbours.
- A corner vertex may be formed between successive shared edges of a planar polygon segment's perimeter.

During these model segment operations, shared edges may be unshared, and redundant model segments may be deleted.

In Chapter 3 the computation of the robot's surface following motion was presented. Each surface following motion attempts to maintain a desired distance between the robot's control point and the surface model. Also the length of each surface following motion equals a set step size. These measures ensure that the control point's sensors remain within sensing range of the surface, and that the control point is prevented from colliding with the surface. Redundancy is used to prevent the rest of the robot from colliding with the surface. A surface following direction forms a component of each surface following motion. Surface following directions direct the robot's control point to non-shared edges, and once at a non-shared edge, direct the control point over the unexposed surface near the edge. The surface following directions are computed by various motion modes:

- Current surface following directs the control point to the current segment via a path of edges on the current surface.
- Current segment following directs the control point to the current edge via the current segment's perimeter.
- Current edge following directs the control point over the unexposed surface near the current edge, by dividing the unexposed surface into slices.
- Gap following aims the control point's sensors into a surface model gap near the current edge, by using a set of target points.

If surface following is making insufficient progress towards the completion of the current surface then the current goal is abandoned, and a new goal is chosen. This guarantees the completion of the current surface.

In Chapter 4 results of surface modelling and surface following for a variety of robots and surfaces using real sensor data were presented. These results demonstrate the effectiveness, when dealing with planar polyhedra, of the surface modelling and surface following techniques presented in this thesis. In developing these techniques the following contributions to robotics have been made:

- Incremental surface modelling techniques that use sensor data obtained from robot-mounted range sensors. These techniques are novel as they deal with sensor data that are quite different from those of traditional image based surface modelling. The sensor data are received as a series of sets of sensor points. The sensor points in these sets have no structure and are fairly sparse. Also surface modelling is performed on-line, as the sensor data are obtained.

- Surface following techniques for the collection of sensor data for use with surface modelling. These techniques make use of the surface model to compute the robot's surface following motion. The control point's sensors are kept within sensing range of the surface while the robot is kept from colliding with the surface. Also the surface following motion's direction is chosen such that as much of the surface as possible is modelled.

- The application of robot-mounted range sensors to the task of three-dimensional surface modelling.

5.2 Suggestions for Future Research

A more reliable means of detecting redundant model segments is required. Although the current test is performed quickly it sometimes incorrectly classifies model segments. As a result, some redundant model segments fail to be deleted, and some non-redundant model segments are deleted. Often those redundant model segments that have failed to be detected have done so because they have formed shared edges.

Thus the signal for performing a redundancy test should not depend solely on the failure of an attempt to form shared edges between two non-neighbouring model segments. Also a more reliable redundancy test should consider more than just the number of sensor points represented by a model segment; the model segment's position and orientation relative to nearby model segments should also be considered. However, increasing the redundancy test's frequency and complexity will also increase its computational expense. This must be considered when designing on-line surface modelling algorithms.

When dealing with curved surfaces, the surface model's accuracy is poor. To accurately model curved surfaces, model segments whose descriptions are of a higher order than points and planar polygons are required. As surface modelling is performed on-line, when choosing the high-order surfaces to use, the computational expense of processing the surface model must be considered, in particular, the cost of computing the distance to the surface model must be considered. The use of model segment data is one reason for the quick performance of the surface modelling step. It allows surface modelling to process each sensor point once and then discard it. Thus high-order surfaces that can be represented using model segment data are desirable. Quadric surfaces are an example of such a surface. The quadric of best fit[1] to a set of points can be determined from matrices constructed from an extended set of model segment data [33,70]. The perimeter of a planar polygon segment is a set of connected edges, where shared edges are defined by the intersection of two planes. Extending this to high-order surfaces, the perimeter becomes a set of connected curves, where shared curves are defined by the intersection of two high-order surfaces. The updating of the surface model, model segment operations and surface following will also need to be adapted for use with high-order surfaces.

[1] However, the mean perpendicular distance is not minimized.

Bibliography

[1] J. K. Aggarwal and C. H. Chien. 3-D structures from 2-D images. In J. L. C. Sanz, editor, *Advances in Machine Vision*, chapter 2, pages 64–159. Springer-Verlag, New York, 1989.

[2] S. Ahmad and C. N. Lee. Shape recovery from robot contour-tracking with force feedback. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 447–452, Cincinnati, Ohio, May 1990. IEEE Computer Society Press.

[3] S. Ahmad and C. N. Lee. Shape recovery from robot contour-tracking with force feedback. *Advanced Robotics*, 5(3):257–273, 1991.

[4] P. K. Allen. Integrating vision and touch for object recognition tasks. *The International Journal of Robotics Research*, 7(6):15–33, December 1988.

[5] P. K. Allen. Mapping haptic exploratory procedures to multiple shape representations. In *IEEE International Conference on Robotics and Automation*, pages 1679–1684, Cincinnati, Ohio, May 1990. IEEE Computer Society Press.

[6] P. K. Allen and P. Michelman. Acquisition and interpretation of 3-D sensor data from touch. In *Workshop on the Interpretation of 3-D Scenes*, pages 33–40, Austin, Texas, November 1989. IEEE Computer Society Press.

[7] C. Bajaj and M. Kim. Generation of configuration space obstacles: Moving algebraic surfaces. *The International Journal of Robotics Research*, 9(1):92–112, February 1990.

[8] A. H. Barr. Superquadrics and angle preserving transforms. *IEEE Computer Graphics and Applications*, 1:11–23, January 1981.

[9] B. G. Baumgart. A polyhedron representation for computer vision. In *AFIPS National Computer Conference*, volume 44, pages 589–596, 1975.

[10] L. Beckley, P. Kovesi, and R. Owens. The use of imaginary actuators in kinematically redundant mechanisms for obstacle avoidance. In *Third National Conference on Robotics*, Melbourne, June 1990.

[11] P. J. Besl. *Surfaces in Range Image Understanding.* Springer-Verlag, New York, 1988.

[12] P. J. Besl and R. C. Jain. Three-dimensional object recognition. *ACM Computing Surveys*, 17(1):75–145, March 1985.

[13] P. J. Besl and R. C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2):167–192, March 1988.

[14] A. Blake, M. Brady, Z. Xie, and A. Zisserman. Visual navigation around curved obstacles. Technical report, Department of Engineering Science, University of Oxford, September 14 1990.

[15] S. H. Boey, B. F. Alexander, and K. C. Ng. Generation of 3-D models by combining active and passive triangulation. In *IEEE International Conference on Image Processing*, volume 2, Singapore, September 1989.

[16] R. M. Bolle and B. C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(1):1–13, January 1991.

[17] R. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Eighth International Joint Conference on Artificial Intelligence*, pages 797–806, August 1983.

[18] C. E. Buckley. A foundation for the "flexible-trajectory" approach to numeric path planning. *The International Journal of Robotics Research*, 8(3):44–64, June 1989.

[19] E. Cheung. *Real-Time Motion Planning for Whole-Sensitive Robot Arm Manipulators*. PhD thesis, Yale University, 1990.

[20] E. Cheung and V. J. Lumelsky. Development of sensitive skin for a 3D robot arm operating in an uncertain environment. In *IEEE Conference on Robotics and Automation*, pages 1056–1061, Scottsdale, AZ, May 1989. IEEE Computer Society Press.

[21] E. Cheung and V. J. Lumelsky. Motion planning for a whole-sensitive robot arm manipulator. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 344–349, Cincinnati, Ohio, May 1990. IEEE Computer Society Press.

[22] C. Chevallereau and W. Khalil. A new method for the solution of the inverse kinematics of redundant robots. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 37–42, Philadelphia, Pennsylvania, April 1988. IEEE Computer Society Press.

[23] R. T. Chin and C. R. Dyer. Model-based recognition in robot vision. *ACM Computing Surveys*, 18(1):67–108, March 1986.

[24] P. Coiffet. *Remote Interaction: Proximity Sensing*, volume 2 of *Robot Technology*, chapter 5, pages 91–100. Prentice-Hall, Inc., London, 1983.

[25] P. Dario and G. Buttazzo. An anthropomorphic robot finger for investigating artificial tactile exploration. *The International Journal of Robotics Research*, 6(3):25–48, 1987.

[26] C. De Medio and G. Oriolo. Robot obstacle avoidance using vortex fields. In *Second International Workshop on Advances in Robot Kinematics*, Linz, Austria, September 1990.

[27] E. D. Dickmanns. Dynamic computer vision for mobile robot control. In *International Symposium and Exposition on Robots*, pages 314–327, Sydney, November 1988.

[28] E. D. Dickmanns and V. Graefe. Applications of dynamic monocular machine vision. Technical report, Universität der Bundeswehr München, July 1988.

[29] A. Elfes. A sonar-based mapping and navigation system. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1151–1156, San Francisco, California, April 1986. IEEE Computer Society Press.

[30] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, June 1987.

[31] B. Espiau and R. Boulic. Collision avoidance for redundant robots with proximity sensors. In O. D. Faugeras and G. Giralt, editors, *Robotics Research 3*, volume 3 of *The MIT Series in Artificial Intelligence*, chapter 6, pages 243–252. The MIT Press, 1986.

[32] O. D. Faugeras and M. Hebert. A 3-D recognition and positioning algorithm using geometric matching between primitive surfaces. In *Eighth International Joint Conference on Artificial Intelligence*, pages 996–1002, Karlsruhe, Germany, August 1983.

[33] O. D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-D objects. *The International Journal of Robotics Research*, 5(3):27–52, 1986.

[34] O. D. Faugeras and M. Hebert. Representation, recognition, and positioning of 3-D shapes from range data. In A. Rosenfeld, editor, *Techniques for 3-D Machine Perception*, pages 13–51. North-Holland, 1986.

[35] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, Chichester, 1979.

[36] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *IEEE International Conference on Robotics*, pages 504–512, March 1984.

[37] R. B. Fisher. Using surfaces and object models to recognise partially obscured objects. In *Eighth International Joint Conference on Artificial Intelligence*, pages 989–995, Karlsruhe, Germany, August 1983.

[38] R. B. Fisher. Geometric constraints from planar surface patch matching. *Image and Vision Computing*, 8(2):148–154, May 1990.

[39] R. B. Fisher. The design of the IMAGINE II scene analysis program. In J. E. W. Mayhew and J. P. Frisby, editors, *3D Model Recognition from Stereoscopic Cues*, pages 239–244. MIT Press, 1991.

[40] R. B. Fisher. SMS: A suggestive modeling system for object recognition. In J. E. W. Mayhew and J. P. Frisby, editors, *3D Model Recognition from Stereoscopic Cues*, pages 221–230. MIT Press, 1991.

[41] A. M. Flynn. Redundant sensors for mobile robot navigation. Technical Report 859, MIT Artificial Intelligence Laboratory, Massachusetts Institute of Technology, October 1985.

[42] A. M. Flynn. Combining sonar and infrared sensors for mobile robot navigation. *The International Journal of Robotics Research*, 7(6):5–14, December 1988.

[43] Y. Ge, R. Owens, and P. Hartmann. Breast volume measurement from multiple views. In *UWA Department of Computer Science Research Conference*, July 1993.

[44] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 1718–1723, Cincinnati, Ohio, May 1990.

[45] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, Reading, Massachusetts, 1992.

[46] L. Gouzènes. Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *The International Journal of Robotics of Research*, 3(4):51–65, 1984.

[47] D. H. Graf and W. R. LaLonde. A neural controller for collision-free movement of general robot manipulators. In *IEEE International Joint Conference on Neural Networks*, pages I–77–I–84, Piscataway, NJ, 1988. IEEE Computer Society Press.

[48] T. Hasegawa. Collision avoidance using characterized descriptions of free space. In Y. Umetani, editor, *International Conference on Advanced Robotics*, pages 69–76, Tokyo, Japan, September 1985. IFS Publications Limited.

[49] B. K. P. Horn. *Robot Vision.* McGraw Hill, Cambridge, Massachusetts, 1985.

[50] K. Ikeuchi. Shape from regular patterns. *Artificial Intelligence*, 22:49–75, 1984.

[51] P. R. M. Jones, G. M. West, D. H. Harris, and J. B. Read. The Loughborough anthropometric shadow scanner (LASS). *Endeavour, New Series*, 13(4):162–168, 1989.

[52] K. Kant and S. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

[53] J. P. Karlen, J. M. Thompson, J. D. Farrell, and H. I. Vold. Reflexive obstacle avoidance for kinematically-redundant manipulators. In *NASA Conference on Space Telerobotics*, pages 25–37, Pasadena, California, January 1989.

[54] K. Kazerounian and Z. Wang. Global versus local optimization in redundancy resolution of robotic manipulators. *The International Journal of Robotics Research*, 7(5):3–12, October 1988.

[55] J. R. Kender. Shape from texture: An aggregation transform that maps a class of textures into surface orientation. In *Sixth International Joint Conference on Artificial Intelligence*, pages 457–480, Tokyo, August 1979.

[56] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ, second edition, 1991.

[57] O. Khatib. Real-time obstacle avoidance for robot manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[58] M. Kirćanski and M. Vukobratović. Contribution to control of redundant robotic manipulators in an environment with obstacles. *The International Journal of Robotics Research*, 5(4):112–119, 1986.

[59] C. A. Klein and B. E. Blaho. Dexterity measures for the design and control of kinematically redundant manipulators. *The International Journal of Robotics Research*, 6(2):72–83, 1987.

[60] C. A. Klein and C.-H. Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Transactions on Systems, Cybernetics, and Man*, SMC-13(3):245–250, March 1983.

[61] W. D. Koenigsberg. Noncontact distance sensor technology. In B. Rooks, editor, *Third International Conference on Robot Vision and Sensory Controls*, pages 371–383, Cambridge, Massachusetts, November 1983. IFS Publications.

[62] P. Kovesi. Imaginary kinematics. In S. Stifter and J. Lenarcic, editors, *The Second International Workshop on Advances in Robot Kinematics*, pages 55–62, Linz, Austria, September 1990. Springer Verlag.

[63] B. H. Krogh. A generalized potential field approach to obstacle avoidance control. In *SME Conference on Robotics Research: The Next Five Years and Beyond*, Bethlehem, Pennsylvania, August 1984.

[64] A. Liégeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(12):868–871, December 1977.

[65] T. Lozano-Pérez. Motion planning for simple robot manipulators. In O. D. Faugeras and G. Giralt, editors, *The Third International Symposium in*

Robotics Research, Artificial Intelligence, pages 133–140, Cambridge, Massachusetts, 1986. MIT Press.

[66] V. J. Lumelsky and E. Cheung. Towards safe real-time robot teleoperation: Automatic whole-sensitive arm collision avoidance frees the operator for global control. In *IEEE Conference on Robotics and Automation*, 1991.

[67] A. A. Maciejewski and C. A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, 4(3):109–117, 1985.

[68] M. Mäntylä. *An Introduction to Solid Modelling.* Computer Science Press, Rockville, Maryland, 1988.

[69] X. Markenscoff and C. H. Papadimitriou. Optimum grip of a polygon. *The International Journal of Robotics Research*, 8(2):17–29, April 1989.

[70] A. D. Marshall and R. R. Martin. *Computer Vision, Models and Inspection.* Robotics and Automated Systems. World Scientific Publishing, 1992.

[71] P. J. McKerrow. *Introduction to Robotics.* Addison Wesley, 1991.

[72] J. K. Myers and G. J. Agin. A supervisory collision-avoidance system for robot controllers. *Robotics World*, 1(1), January 1983.

[73] W. S. Newman and N. Hogan. High speed robot control and obstacle avoidance using dynamic potential functions. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 14–24, Raleigh, NC, March 1987.

[74] V. Nguyen. Constructing force closure grasps. *The International Journal of Robotics Research*, 7(3):3–16, June 1988.

[75] Y. Ohta, K. Maenobu, and T. Sakai. Obtaining surface orientation from texels under perspective projection. In *Seventh International Joint Conference on Artificial Intelligence*, volume II, pages 746–751, Vancouver, Canada, August 1981.

[76] M. Oshima and Y. Shirai. A scene description method using three-dimensional information. *Pattern Recognition*, 11:9–17, 1979.

[77] M. Oshima and Y. Shirai. Object recognition using three-dimensional information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(4):353–361, July 1983.

[78] B. Paden, A. Mees, and M. Fisher. Path planning using a Jacobian-based freespace generation algorithm. In *IEEE Conference on Robotics and Automation*, pages 1732–1737. IEEE Computer Society Press, May 1989.

[79] Y. C. Park and G. Starr. Grasp synthesis of polygonal objects using a three-fingered hand. *The International Journal of Robotics Research*, 11(3):163–184, June 1992.

[80] R. Paul. Manipulator Cartesian path control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9(11):702–711, November 1979.

[81] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 6:559–572, 1901.

[82] C. J. Pudney. Surface following for manipulators with proximity sensors. In *UWA Department of Computer Science Research Conference*, Yanchep, Australia, July 1991.

[83] C. J. Pudney. Surface following and modelling for planar n-link manipulator arms equipped with proximity sensors. In *Third Annual Conference of AI, Simulation and Planning in High Autonomy Systems*, pages 54–61, Perth, Australia, July 1992.

[84] C. J. Pudney. Surface following and modelling for planar robots. In *UWA Department of Computer Science Research Conference*, Yanchep, Australia, July 1992. Also UWA Dept. of Computer Science Technical Report 92/4.

[85] C. J. Pudney. Surface following for robots operating in 3D environments. In *UWA Department of Computer Science Research Conference*, Yanchep, Australia, July 1993.

[86] C. J. Pudney. Surface modelling for robots equipped with range sensors. In *First Australian and New Zealand Conference on Intelligent Information Systems*, pages 79–83, Perth, Western Australia, December 1993. IEEE Computer Society Press.

[87] C. J. Pudney. Surface modelling for surface following robots. In G. Gupta, editor, *17th Annual Computer Science Conference*, volume A, pages 43–54, Christchurch, New Zealand, January 1994.

[88] C. R. Rao and S. K. Mitra. *Generalized Inverse of Matrices and its Applications*. Wiley, New York, 1971.

[89] Z. Shiller and S. Dubowsky. Robot path planning with obstacles, actuator, gripper, and payload constraints. *The International Journal of Robotics Research*, 8(6):3–18, December 1989.

[90] Y. Shirai. *Three-Dimensional Computer Vision*. Springer-Verlag, Berlin, 1987.

[91] S. A. Stansfield. Primitives, features, and exploratory procedures: Building a robot tactile perception system. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1274–1279, San Francisco, April 1986.

[92] S. A. Stansfield. A robotic perceptual system utilizing passive vision and active touch. *The International Journal of Robotics Research*, 7(6):128–161, December 1988.

[93] S. A. Stansfield. A haptic system for a multifingered hand. In *IEEE International Conference on Robotics and Automation*, pages 658–664, Sacramento, California, April 1991. IEEE Computer Society Press.

[94] L. Tarassenko and A. Blake. Analogue computation of collision-free paths. Technical report, Department of Engineering Science, University of Oxford, Oxford, U.K., September 1990.

[95] J. C. Trinkle, J. M. Abel, and R. P. Paul. An investigation of frictionless enveloping grasping in the plane. *The International Journal of Robotics Research*, 7(3):33–51, June 1988.

[96] J. C. Trinkle and R. P. Paul. Planning for dexterous manipulation with sliding contacts. *The International Journal of Robotics Research*, 9(3):24–48, June 1990.

[97] M. W. Vannier, T. Pilgram, G. Bhatia, B. Brunsden, and P. Commean. Facial surface scanner. *IEEE Computer Graphics and Applications*, pages 73–80, November 1991.

[98] C. W. Warren, J. C. Danos, and B. W. Mooring. An approach to manipulator path planning. *The International Journal of Robotics Research*, 8(5):87–95, October 1989.

[99] D. E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, MMS-10(2):47–53, June 1969.

[100] R. J. Woodham. Photometric stereo: A reflectance map technique for determining surface orientation from image intensity. In *SPIE 22nd Annual Technical Symposium: Image Understanding Systems and Industrial Applications*, volume 155, pages 136–143, San Diego, California, August 1978.

Appendix A

Derivation of the Reconfiguration Vector

This appendix presents a derivation of the reconfiguration vector used to prevent collisions between the rest of the robot and the surface. The reconfiguration vector is obtained from the negative gradient of an objective function (4) to be optimized:

$$\mathbf{z} = - \bigtriangledown \mathcal{H}(\Theta) = -\frac{d_0}{g} \sum_{i=1}^{n} \begin{cases} \bigtriangledown (1 - \frac{d_i}{d_0})^g & d_i < d_0 \\ 0 & d_i \geq d_0 \end{cases} .$$

Thus the components of the reconfiguration vector are given by:

$$z_k = -\frac{d_0}{g} \sum_{i=k}^{n} \begin{cases} \frac{\partial (1 - \frac{d_i}{d_0})^g}{\partial \theta_k} & d_i < d_0 \\ 0 & d_i \geq d_0 \end{cases} .$$

Note that the summation is only over values $k, \ldots, n$ as the first $k-1$ links are unaffected by changes of the kth actuator. Considering the $d_i < d_0$ case:

$$\begin{aligned} z_k &= -\frac{d_0}{g} \sum_{i=k}^{n} \frac{\partial (1 - \frac{d_i}{d_0})^g}{\partial \theta_k} \\ &= -d_0 \sum_{i=k}^{n} (1 - \frac{d_i}{d_0})^{g-1} \frac{\partial (1 - \frac{d_i}{d_0})}{\partial \theta_k} \\ &= \sum_{i=k}^{n} (1 - \frac{d_i}{d_0})^{g-1} \frac{\partial d_i}{\partial \theta_k} \qquad (7) \\ \frac{\partial d_i}{\partial \theta_k} &= \frac{1}{2} \frac{\partial (\mathbf{l}_i - \mathbf{s}_i).(\mathbf{l}_i - \mathbf{s}_i)}{\partial \theta_k} \\ &= (\mathbf{l}_i - \mathbf{s}_i). \frac{\partial (\mathbf{l}_i - \mathbf{s}_i)}{\partial \theta_k} \end{aligned}$$

$$= (\mathbf{l}_i - \mathbf{s}_i).\frac{\partial \mathbf{l}_i}{\partial \theta_k} \tag{8}$$

$$\frac{\partial \mathbf{l}_i}{\partial \theta_k} = \begin{cases} \hat{\mathbf{a}}_k \times (\mathbf{l}_i - \mathbf{q}_k) & k\text{th joint revolute} \\ \hat{\mathbf{a}}_k & k\text{th joint prismatic} \end{cases}, \tag{9}$$

where $\hat{\mathbf{a}}_k$ is the unit vector that describes the kth actuator's axis of motion, and $\mathbf{q}_k$ is the kth actuator's position in space. Combining (7), (8), and (9) yields:

$$z_k = \sum_{i=k}^{n} \begin{cases} (1 - \frac{d_i}{d_0})^{g-1}(\mathbf{l}_i - \mathbf{s}_i).\mathbf{r}_{ik} & d_i < d_0 \\ 0 & d_i \geq d_0 \end{cases},$$

$$\mathbf{r}_{ik} = \begin{cases} \hat{\mathbf{a}}_k \times (\mathbf{l}_i - \mathbf{q}_k) & k\text{th joint revolute} \\ \hat{\mathbf{a}}_k & k\text{th joint prismatic} \end{cases}.$$

Note that $\mathbf{l}_i, \mathbf{s}_i, \hat{\mathbf{a}}_k$ and $\mathbf{q}_k$ are defined relative to a global coordinate frame.

Appendix B

An Objective Function for use with Proximity Switches

This appendix presents an objective function for use with robots whose links are covered with proximity switches. A proximity switch produces a binary signal: object present/no object present. This provides no information regarding the distance and direction to the object[1] thus making it difficult to determine the object point that is closest to the proximity switch. If it is assumed that when a proximity switch detects an object, the object point that is closest to the switch is at the centre of the switch's sensing field then an objective function analogous to (4) can be used:

$$\begin{aligned} \mathcal{H}(\Theta) &= \frac{d_0}{g} \sum_{i=1}^{p} \begin{cases} (1 - \frac{d_i}{d_0})^g & i\text{th switch: object present} \\ 0 & i\text{th switch: no object present} \end{cases}, \\ d_i &= \frac{1}{2}\|\mathbf{p}_i - \mathbf{c}_i\|^2 = \frac{1}{2}(\mathbf{p}_i - \mathbf{c}_i).(\mathbf{p}_i - \mathbf{c}_i), \end{aligned}$$

where g is a positive integer, p is the number of proximity switches the robot has, $\sqrt{2d_0}$ is the sensing range of the proximity switches, $\mathbf{p}_i$ is the ith proximity switch's position in space, and $\mathbf{c}_i$ is centre of the ith proximity switch's sensing field. Note that each $\mathbf{p}_i$ is a function of actuator positions θ_1 through θ_j, where the ith proximity switch is mounted on the jth link. Thus the components of the

[1] Distance and direction information can be obtained from some proximity switches but its accuracy varies greatly with surface properties.

reconfiguration vector are given by:

$$z_k = \sum_i \begin{cases} (1 - \frac{d_i}{d_0})^{g-1}(\mathbf{p}_i - \mathbf{c}_i).\mathbf{r}_{ik} & i\text{th switch: object present} \\ 0 & i\text{th switch: no object present} \end{cases},$$

$$\mathbf{r}_{ik} = \begin{cases} \hat{\mathbf{a}}_k \times (\mathbf{p}_i - \mathbf{q}_k) & k\text{th joint revolute} \\ \hat{\mathbf{a}}_k & k\text{th joint prismatic} \end{cases},$$

where the summation is over the proximity switches mounted on links k through n, $\hat{\mathbf{a}}_k$ is the unit vector that describes the kth actuator's axis of motion, and $\mathbf{q}_k$ is the kth actuator's position in space. Note that $\mathbf{c}_i$ is fixed relative to $\mathbf{p}_i$, so $(1 - \frac{d_i}{d_0})^{g-1}(\mathbf{p}_i - \mathbf{c}_i)$ is a constant. Note also that $\mathbf{p}_i, \mathbf{c}_i, \hat{\mathbf{a}}_k$ and $\mathbf{q}_k$ are defined relative to a global coordinate frame.

Appendix C

Surface Following for Robots Operating in 3D Environments

In *UWA Deparment of Computer Science Research Conference*, Yanchep, Australia, July 1993.

Appendix D

Surface Modelling for Robots Equipped with Range Sensors

In *First Australian and New Zealand Conference on Intelligent Information Systems*, pages 79–83, Perth, Australia, December 1993. IEEE Computer Society Press.

Appendix E

Surface Modelling for Surface Following Robots

In G. Gupta, editor, *17th Annual Computer Science Conference*, volume A, pages 43–54, Christchurch, New Zealand, January 1994.