Computing global configuration-space maps using multidimensional set-theoretic modelling

Wise, Kevin D.

Award date:
2000

Awarding institution:
University of Bath

COMPUTING GLOBAL CONFIGURATION-SPACE MAPS USING MULTIDIMENSIONAL SET-THEORETIC MODELLING

Submitted by Kevin D Wise
for the degree of
Doctor of Philosophy
of the University of Bath
2000

10 FEB 2000

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

This thesis describes two ways that multidimensional set-theoretic modelling can be used to compute global maps of the C-space for a system of rigid bodies.

The first combines workspace dimensions and degrees of freedom to construct a multidimensional model which contains information regarding every interaction which occurs as the moving object (or system of objects) exercises its degrees of freedom. When interference regions within this multidimensional model are projected into the C-space they map out the C-space obstacles. Five heuristic algorithms are presented which compute an approximate C-space map via this projection. All have been successfully implemented and have been shown to be inherently general.

The second method of C-space mapmaking presented is the precise representation of C-space obstacles by formulating contact surfaces analytically. Algorithms are described which exploit several properties of multidimensional set-theoretic modelling which suit it to this task. These algorithms have been implemented, resulting in a mapmaker which has been shown to compute valid precise global C-space maps for three dimensional objects with six degrees of freedom. Moroever, the algorithm for a single moving object extends in a straightforward manner to handle multiple independent moving objects and to manipulators. A mapmaker has therefore been implemented which has computed precise global C-space maps for up to six independent objects with eighteen degrees of freedom, and precise global C-space maps for a robot manipulator with up to six revolute joints.

A number of possible future work directions are outlined, including the combination of the two approaches into a hybrid C-space mapmaker.

Acknowledgements

I would like to acknowledge that Adrian is laid-back, Dave is cool, my mum is great and my dad is missed by everyone who knew him.

I would also like to thank EPSRC for funding this research, Southco Ltd. for their industrial input, and G-mod for their support and friendship.

Finally, I would like to thank Ann for, er, everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 An anecdote

In January 1865, English aristocrat George Beecham, known for his sense of adventure and fearless disposition and inspired by the exploits of David Livingstone, set off from Southampton on a voyage to Africa taking only a big hat, two spare shirts and some smalls. Beecham shrugged off suggestions that he should take a map, declaring that as bold a man as he could deal with anything that he could come across and besides, he continued, so much more could be achieved in life if you concentrated on where you were, rather than where you were going. Less than a week into his trek across the continent, however, he stumbled across quicksand into which, despite his protestations, he sank without trace.

The moral of this story is that no matter how much fun it is to concentrate on where you are, having a map of where you can and cannot go will allow you to plan ahead and avoid areas that could get you in all sorts of trouble. This thesis describes methods which, for a given system of rigid objects, will compute a global *map* of the configuration space of that system—that is, a representation of the set of positions and orientations that system of objects can adopt, identifying which of those configurations are prohibited (since they would result in overlap with one or more other objects) and which are safe.

1.2 The configuration-space approach to spatial planning

Few concepts have played a greater role in the study of rigid-body kinematics than the *configuration space* approach to spatial planning, as formally defined by Lozano-Pérez 1983 [63]. A *configuration* of an object or group of objects is a set of parameters sufficient to position every point in the object(s) in space. Thus, for a rigid three-dimensional body with a full six degrees of freedom, a configuration is a six-tuple which specifies the position and orientation of the body relative to (arbitrary) reference values[1]. Similarly, the configuration of a system of rigid bodies with a total of n degrees of freedom may be defined by an an n-tuple.

The *configuration space* (or *C-space*) for a system of rigid objects is the space of all its possible configurations and is therefore an n-dimensional space bounded by upper and lower limits on each of the degrees of freedom. Since two solid objects cannot overlap, configurations of such a system can fall into three categories: *prohibited* (or *forbidden*) configurations, in which objects would overlap; *safe* (or *free*) configurations in which no overlap occurs; and *contact* configurations in which two or more objects touch each other in one or more places.

Throughout this thesis I use the term *C-space map* to refer to a geometric model which partitions the C-space into these three categories. The prohibited regions in such a map are commonly referred to as *C-space obstacles* and the safe regions as *free-space* (although this term is also sometimes used in the literature to refer to the complement of the obstacles in the workspace). The boundary between the C-space obstacles and the free-space is referred to as the *contact surface.*

On the subject of terminology, throughout this thesis I use the term '*nomad*' to refer to a rigid-body object which has degrees of freedom which are dimensions of the configuration-space; I use the term '*obstacle*' to refer to an object which a nomad must avoid overlapping with, and I refer to the two- or three-dimensional space in which the nomad and obstacles exist as the *workspace.* The

[1] In fact, the configuration of a three-dimensional body might by parametrised by *seven* values since its orientation might be parameterised by a quaternion which is a four-tuple. For simplicity, throughout this thesis I assume that the configuration of an object with n degrees of freedom is parameterised by n values, which is the minimum.

notation $\mathcal{CSOBS}$ refers to the set of prohibited configurations, and more specifically, $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{NOM}}$ is used to refer to the C-space obstacle which obstacle OBS causes to nomad NOM. The notation $\mathcal{FREESPACE}$ refers to the set of safe configurations.

The relationship between workspace and C-space is illustrated in Figures 1.1 and 1.2—the first considers the simple case of a two-dimensional translating object, whilst the second shows an example of C-space mapping for a manipulator, after Siméon [91].

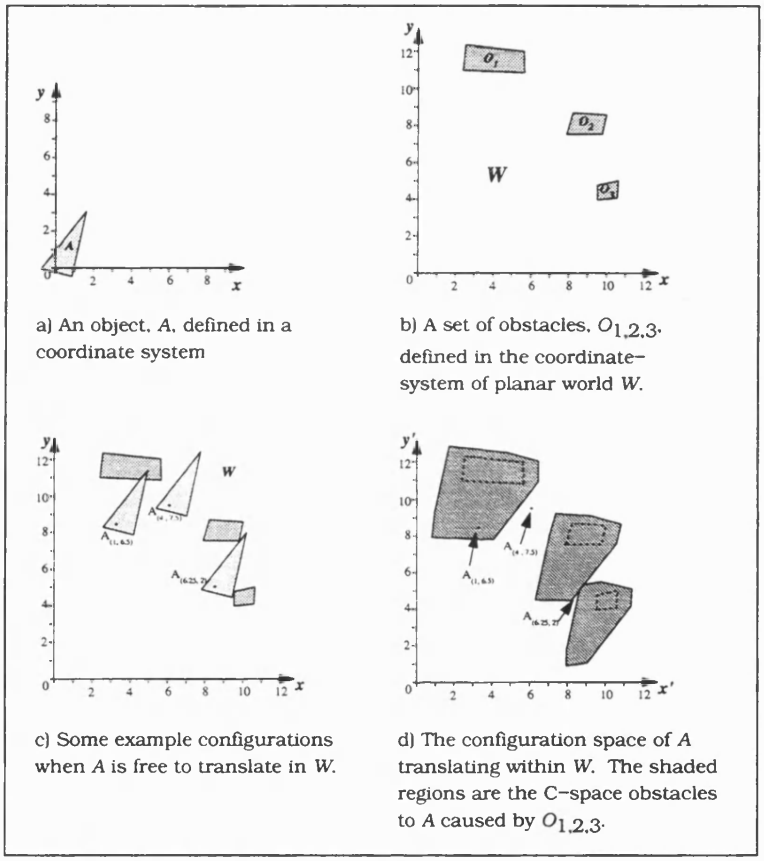
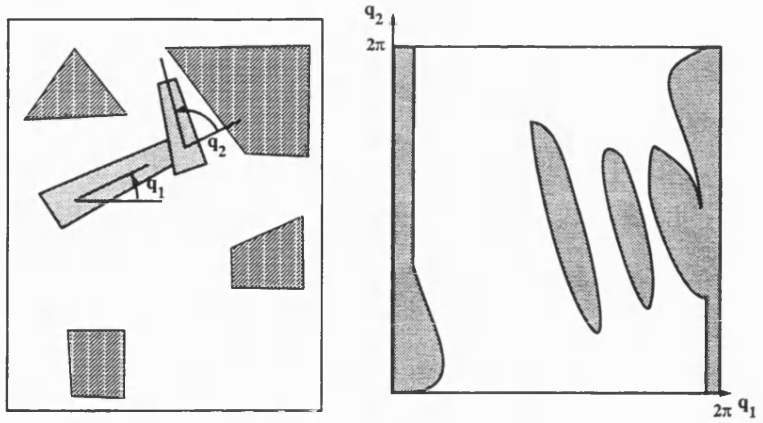


Figure 1.1: **An illustration of C-space mapping.**

Generating a C-space map—that is, transforming a problem from the workspace into the configuration space—is a far from trivial task for all but the most elementary problems. Indeed, in the worst case the amount of information contained in a C-space map increases exponentially with the number of degrees of freedom (for an in-depth study of the complexity of C-space mapping and related issues see for example Reif 1979 [82], Hopcroft, Schwartz and Sharir 1984 [41], Canny



(a) A polygonal model of a 2–DOF manipulator in an environment with obstacles.

(b) The obstacles in the 2–D configuration space.

Figure 1.2: **An illustration of C-space mapping for a manipulator, after Siméon [91].**

1988 [15], Cesati and Wareham 1995 [16]). A second disadvantage of the C-space approach is that it relies on an *a priori* knowledge of the geometry of the objects involved. However there is a strong motivation for performing the C-space mapping step whenever possible: a continuous path of the system of rigid objects through the 2- or 3-dimensional world corresponds to a path for a 0-dimensional point through the C-space map. This means that the need for 2- or 3-dimensional collision detection has been removed and the path planning problem is reduced to finding a curve which connects the initial and goal configurations without entering a prohibited region.

For this reason, C-space maps are most commonly used in robotics path planning where optimal global paths are achieved by arranging a decomposition of the free-space (and/or contact surface) into a graph and then applying a graph-search algorithm such as the A*(Hart, Nilsson and Raphael 1968 [37]). This method is not restricted to the case of a single robot in a static environment—C-space techniques have been developed to model both mobile robots and manipulators in environments which change predictably (for example Fujimura and Samet 1993 [31]) and systems where multiple nomads share an environment (for example Parsons and Canny 1990 [76]).

Beyond robotics, C-space maps have also been applied successfully in the areas of packing and nesting (for example Adamowicz and Albano 1976 [2]), automated

assembly planning(for example Schweikard and Wilson 1995 [88]) and mechanism analysis. In the latter area, researchers have exploited the effectiveness of C-space maps to capture the kinematic constraints imposed upon a set of rigid objects by their geometry and to describe every change of contact between the components of a mechanism. C-space maps have thus been seized upon as a key link between shape and behaviour and have been used successfully for reasoning about kinematic behaviour (for example Faltings 1987 [28]), kinematic simulation (for example Joskowicz and Sacks 1993 [85]), design classification and retrieval (for example Murakami and Gossard 1992 [70]) and automated innovative design (for example Subramanian and Wang 1995 [95]).

Applications of C-space techniques beyond the realm of a single robot in a static environment are discussed in Wise and Bowyer [106].

1.3 Multidimensional geometry

Throughout this thesis I refer to *multidimensional* objects as defined :

> "**Multidimensional** ... *adj.* of or involving more than three dimensions"
>
> *The Concise Oxford Dictionary, Ninth Edition*

Note that this contrasts with the definition used by some authors (for example Regli et al. [81], Vaněček [99]), which refers to representations of objects of mixed dimensionality where users are able to work with entities of different dimensionalities associated with a single object (for example a model of a printed circuit board may contain user-editable objects of both two and three dimensions).

Despite the development of various tools (see for example Inselberg and Dimsdale [45]) objects with more than three dimensions are astonishingly difficult to visualise. Whereas three-dimensional solids have two-dimensional surfaces which are joined by one-dimensional edges which meet at zero-dimensional vertices, an n-dimensional solid has $(n-1)$-dimensional *hypersurfaces* which are joined at

$(n-2)$-dimensional hypersurfaces which meet at $(n-3)$ dimensional hypersurfaces... and so on until one-dimensional edges are reached which meet at zero-dimensional vertices. Topological considerations clearly become complicated.

Perhaps surprisingly, it is not necessary to read a weighty treatise on multidimensional geometry in order to understand the research described in this thesis—so for that the interested reader is referred to texts such as Abbot [1], Woods [108], Kendall [51] and Coxeter [20]. As explained in the next section, certain properties of set-theoretic modelling and the way we implement it at Bath make the leap from three-dimensions to many dimensions straightforward, and little multidimensional visualisation is required. In particular, at this stage it is sufficient to understand that an n-dimensional space is one which has n perpendicular axes and that a point in such a space may move an arbitrary distance parallel to any one of those axes without changing its coordinate in any other dimension.

1.4 Set-theoretic modelling

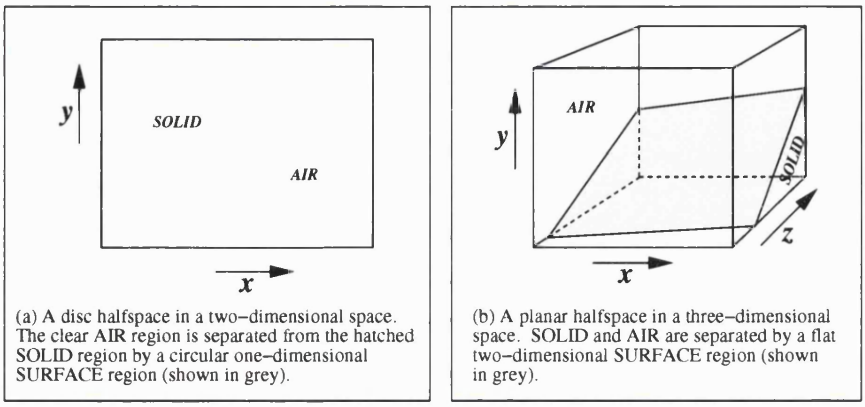
Since its emergence in the early 1980's, the field of Geometric Modelling has been dominated by two very different competing data structures—*boundary representation* (commonly referred to as *B-rep*) and *set-theoretic modelling* (also known as *constructive solid geometry* or *CSG*).

B-rep, the prominent structure in both industry and research institutions, explicitly stores a description of the surface of an object and uses a convention such as 'surface normals point to the exterior' to define which side of each surface is 'solid'. The B-rep structure has two parts (Hoffman [40]):

- A representation of the *topology* of the object, typically a graph structure which stores the connectivity between vertices, edges and faces.
- A representation of the *geometry* of the object, which explicitly locates each vertex, edge and face in space. Edges and faces are typically stored using parametric curves and surfaces.

The other scheme, set-theoretic modelling, represents objects as a set-theoretic

expression. This typically takes the form of a tree with boolean operators (most commonly union, intersection and complement) as the internal nodes and *primitives* (building block shapes) as the leaves. Primitives are *halfspaces*—they divide the space in which they are embedded into two regions (*solid* and *air*) with a third zero-thickness (*surface*) region dividing them. This is illustrated in Figure 1.3 whilst Figure 1.4 shows some of the primitives typically used by a set-theoretic modeller.



(a) A disc halfspace in a two–dimensional space. The clear AIR region is separated from the hatched SOLID region by a circular one–dimensional SURFACE region (shown in grey).

(b) A planar halfspace in a three–dimensional space. SOLID and AIR are separated by a flat two–dimensional SURFACE region (shown in grey).

Figure 1.3: **Illustrations of halfspaces.**

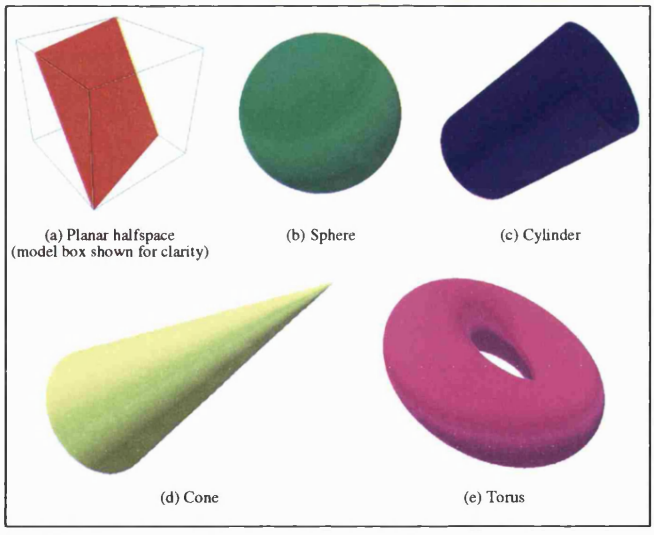


Figure 1.4: **Some typical primitives used by a set-theoretic modeller.**

In contrast to B-rep, the set-theoretic representation does not contain an explicit representation of the surface of the object: the primitives are usually defined using implicit polynomials (see Section 3.2.5) and the surfaces, edges and vertices which result from the boolean operations are not evaluated.

The B-rep and CSG schemes both have strengths and weaknesses. In particular, the strengths of B-rep are that:

- The explicit nature of B-rep makes it possible to generate images directly, whilst an intermediate step is required to locate the surface of an unevaluated CSG tree. This is arguably the primary reason for the dominance of B-rep.
- Popular parametric representations for sculptured surfaces (such as Bézier surfaces and NURBS[2]) are easily incorporated into B-rep, whilst their lack of a straightforward inside/outside and their parametrically bounded nature make their incorporation into CSG non-trivial (although strides in this direction have been made—see Berchtold [9]).
- The local (that is, finite and explicitly spatially located) nature of B-rep vertices, edges and faces makes it easy to implement 'tweaking' operations which allow the user to make small changes to the object in real time via a graphical user interface.

On the other hand:

- Although surfaces and edges are represented implicitly in set-theoretic modelling, *solidity* is represented much more explicitly than it is in B-rep. Boolean operations on point sets are mathematically well-defined and many of the topological issues which B-rep has to deal with are sidestepped. For this reason, the set-theoretic scheme is inherently more numerically stable.
- Testing whether a point is in a solid, air or surface region of a model—particularly important in configuration-space analysis for example—is faster and more accurate in a set-theoretic modeller. This property also makes volume information (and thus mass properties) more readily available (using Monte-Carlo methods for example).
- Since a CSG implementation does not require connectivity information or the explicit representation of complicated curves and surfaces, models typically require less memory than the B-rep equivalent.

[2]Non-Uniform Rational B-Splines.

The research described in this thesis exploits one specific property of set-theoretic modelling:

> The notational structure of an object defined set-theoretically is *independent* of the dimensionality of that object.

Since an n-dimensional hypersolid contains 0-D vertices, 1-D edges, 2-D faces, 3-D solids ... $(n-1)$-D hypersolids, introducing extra dimensions to the B-rep structure quickly makes the topological representation intractable. However, introducing extra dimensions to the set-theoretic scheme requires only that additional variables are introduced to the implicit polynomials which define the primitives—the set-theoretic tree is independent of dimensionality. This makes it tractable to implement a dimension-independent set-theoretic modelling kernel—indeed, such an implementation lies at the heart of this research and is described in Section 3.4.

1.5 Computing global C-space maps using multidimensional set-theoretic modelling

This thesis investigates the application of multidimensional set-theoretic modelling to the problem of generating global configuration-space maps. Two implemented approaches are described: The first is an inherently general method for computing approximate C-space maps, based on a novel idea originally applied to feature recognition; the second approach computes C-space maps precisely using analytical techniques which exploit several properties of set-theoretic modelling. The potential for these two approaches working together in a hybrid solution is discussed in Future Work (Chapter 9).

Note that the focus is specifically on the *computation* of global configuration space maps. Although Chapter 9 touches upon the operations supported by the C-space maps generated by my algorithms, and how the maps could ultimately be used, the development of any practical applications is the beyond the scope of the current work. It is also worth noting that the aim is to demonstrate the application of multidimensional set-theoretic modelling to the area of C-space

mapping in a wide range of contexts—correspondingly, the focus is on breadth, rather than depth.

Chapter 2

A survey of configuration-space techniques for a single nomad in a static environment

2.1 Introduction

Over the past twenty years an enormous amount of research has gone into the mapping from workspace to C-space, resulting in a raft of techniques which can be combined to suit specific applications. Some of these are described in textbooks on robotics (for example Latombe 1993 [53]) and in surveys on the wide field of motion planning (for example Hwang and Ahuja 1992 [44], Hwang 1995 [43]). However, the literature does not appear to contain a comprehensive examination of C-space techniques available to those interested in using a C-space method for their chosen application.

This chapter (which will appear in a slightly different form in Wise and Bowyer [104]) aims to provide an overview of the techniques developed to date to map the global C-space for two of the most fundamental classes of problem—a single nomad or a manipulator in a static environment. Papers concerned with the use, as opposed to the generation, of C-space maps (for example Cheng and Cheng 1996 [19]) are not mentioned hereafter, and likewise for papers which map either a small subset of the C-space (for example Kavraki, Kolountzakis and Latombe 1996 [50]) or

an alternative space (for example Schwartz and Sharir 1983 [87]). For a treatise of the mathematical structure of C-space (including topological and differential properties) and a detailed discussion of algebraic and geometric properties relating to the mapping of C-space obstacles, see Latombe 1993 [53].

The rest of the chapter is organised as follows: Section 2.2 examines issues concerning how the workspace modelled, including a look at how approximations can be used to ease the C-space mapping, whilst Section 2.3 describes the range of schemes used to represent a C-space map. Sections 2.4 and 2.5 discuss the key techniques used to generate a C-space map for a single nomad and manipulators, focusing on the range of techniques rather than on individual papers. In contrast, Section 2.6 contains tables which list some fifty mapmaking papers, classifying each according to the criteria identified in Sections 2.2 and 2.3. Finally, Section 2.7 draws some conclusions from the findings of the survey.

2.2 Modelling the nomad and its environment

2.2.1 Geometry of the objects involved

Three key aspects of the geometry of objects in the workspace affect the difficulty of the C-space mapping:

Dimensionality Clearly, three-dimensional problems are more difficult than two-dimensional problems. In particular, new types of contact between objects are introduced—for example, edge-edge contacts must be considered between polyhedra whilst between polygons such contacts are singularities which do not need attention. Also increasing the dimensionality of the problem affects the potential dimensionality of the C-space.

Convexity For most algorithms, non-convex objects are much more difficult to handle than convex objects. Analytical representation of the contact surface is harder to formulate because contact between two non-convex objects

may occur simultaneously at multiple discrete points. Also distance computations (commonly used by divide-and-classify approaches) are far less complicated between convex objects. The general problem of decomposing an arbitrary solid object into convex parts is very difficult (see Bajaj 1988 [6], Requicha 1983 [83]) but is possible for polyhedra and other practical cases. When this can be done the C-space obstacles can be generated piecewise by exploiting the fact that C-space mapping is distributive over set union, that is:

$$\mathcal{CSOBS}^{\mathsf{C}}_{\mathsf{A}\cup\mathsf{B}} \equiv \mathcal{CSOBS}^{\mathsf{C}}_{\mathsf{A}} \cup \mathcal{CSOBS}^{\mathsf{C}}_{\mathsf{B}}$$

It is also worth noting that if two objects are convex then the C-space obstacle which one causes to the other will itself be convex.

Surface algebraic degree Increasing the algebraic degree of the surfaces of the input models increases the difficulty of the C-space mapping for both analytical methods and most divide-and-classify approaches. The result, as will be seen in Section 2.6, is that the majority of mapmaking systems restrict their interest to polygonal or polyhedral input models. For most robotics applications this is practical since objects should not get close enough for their precise geometry to be important, but this is not true in other applications such as mechanism analysis.

As a compromise between polytopes and arbitrary geometry, some mapmakers handle *generalised polytopes* which, in addition to flat surfaces, contain simple curved shapes —circular arcs in two dimensions and parts of spheres or cylinders in three.

2.2.2 Representation schemes

By far the most common representation scheme for the workspace is the *boundary representation* (B-rep), whereby the surfaces of the objects are represented by lines or patches, which may be parametric (for a description of this and other solid-modelling representations see, for example, Woodwark 1986 [109]). Such a representation can be used as input for analytical, divide-and-classify or hybrid algorithms (see Section 2.3).

The other common representation scheme is discretisation into axially-aligned rectangloid cells (pixels in two dimensions or voxels in three), which may employ a tree-structure such as the binary- or 2^n-tree. This has the disadvantage that it limits the mapmaker to discrete methods but has the advantage that the objects' representations can sometimes be obtained in real-time directly from sensor data.

Despite the dimension-independent property of set-theoretic modelling, my system (Wise and Bowyer 1996 [105]) is the only mapmaker surveyed to exploit that representation.

Divide-and-classify mapmakers commonly employ a hierarchical approximation method which could be implemented using any of the above methods. Objects are represented by a tree in which each level is a successively simple and more conservative approximation of the real shape (as illustrated in Figure 2.1); tests for intersection can then begin by doing fast tests on the simple representations and only progress up to the slower more accurate ones if necessary. Lozano-Pérez credits Marr and Nishihara 1977 [68] with this idea.

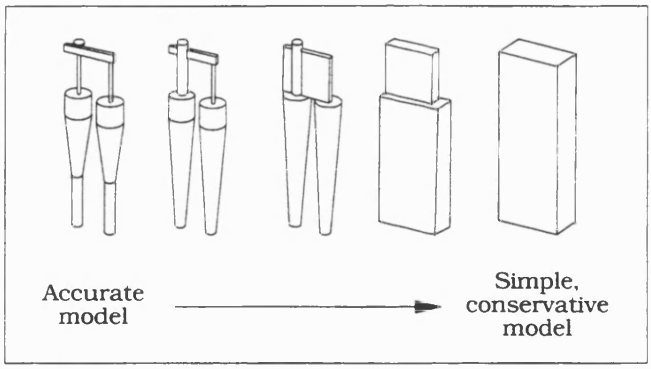


Figure 2.1: **An illustration of hierarchical approximation, after Faverjon and Tournassoud 1988 [30].**

2.2.3 Introduction of approximation

Performing the C-space mapping precisely is so difficult that a key question is whether or not to ease the burden by introducing approximation at the object-modelling or C-space mapping stages. The classification tables in Section 2.6 use the following definitions for the four possible strategies:

Precise object models, precise C-space map The objects are modelled as shapes more complicated than generalised polytopes using a 'continuous' scheme such as B-rep or CSG; precise analytical representations of the C-space obstacles are obtained (e. g. Bajaj 1990 [7]).

Precise object models, approximate C-space map The objects are modelled as shapes more complicated than generalised polytopes using a 'continuous' scheme such as B-rep or CSG; the C-space mapping stage makes approximations such as using axially aligned discrete cells (e. g. Bellier, Laugier, Mazer and Troccaz 1992 [8]).

Approximate object models, precise C-space map The objects are modelled approximately (for example as polytopes, generalised polytopes or discretised cells); the C-space mapping makes no further approximations—either precise analytical representations of the C-space obstacles are obtained (e. g. Avnaim, Boissonnat and Faverjon 1988 [4]) or discrete methods are used without introducing significant further approximations (e. g. Kavraki 1993 [48]).

Approximate object models, Approximate C-space map The objects are modelled approximately (for example as polytopes, generalised polytopes or discretised cells); the C-space mapping introduces further approximations such as using slice-projections[1] (e. g. Lozano-Pèrez and Wesley 1979 [62]) or transferring to a different discretised space (e. g. Tso and Liu 1993 [97]).

2.3 C-space map representation

Central to any C-space mapping system is the method it uses to represent the map. The various choices of representation scheme can be arranged in a taxonomy as shown in Figure 2.2. As illustrated, the methods—any of which can be used to decompose the C-space into a set of connected regions which can subsequently be arranged in a graph—can be split into three main sub-groups: *analytical* methods, *division-and-classification* methods and *hybrid* methods. Every scheme has strengths and weaknesses and the choice of scheme will depend on a number of application-specific factors such as the available data, the number and types

[1]Slice-projections are described in Section 2.4.2.

of degrees of freedom, the relative importance of speed or accuracy, and the way in which the final map will be used.

Note that I use the term *divide-and-classify* to refer to algorithms which decompose the C-space into discrete cells, each of which is then classified as *safe*, *prohibited* or *contact*. Divide-and-classify is therefore a specific example of the more general approach of *divide-and-conquer* (discussed later), which breaks a complicated problem into a (typically large) number of simpler problems.

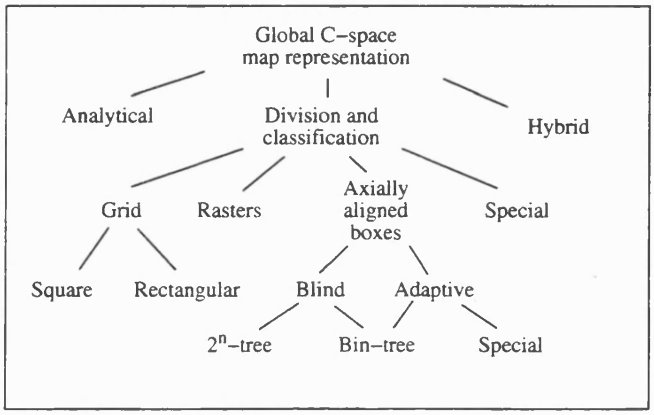


Figure 2.2: **A taxonomy of C-space representation schemes.**

2.3.1 Analytical methods

Analytical methods are characterised as using algebraic techniques to obtain a precise representation of the C-space obstacles, often using the same representation for the C-space obstacles as for the objects in the workspace. Within this survey they are subdivided into two types—*obstacle growing* methods and *contact surface* methods.

Obstacle-growing

In his seminal 1983 paper [63] Lozano-Pérez demonstrated how, for a nomad translating without rotation, the C-space obstacle could be generated precisely by growing the workspace obstacles using Minkowski point-set operations. Minkowski

sum, monadic minus and difference are defined on sets of points (equivalently vectors) in $\Re^n$, A, B as follows:

$$A \oplus B = \{a + b | a \in A, b \in B\}$$

$$\ominus B = \{-b | b \in B\}$$

$$A \ominus B = \{a - b | a \in A, b \in B\} \equiv A \oplus (\ominus B)$$

These operations are illustrated in Figure 2.3.

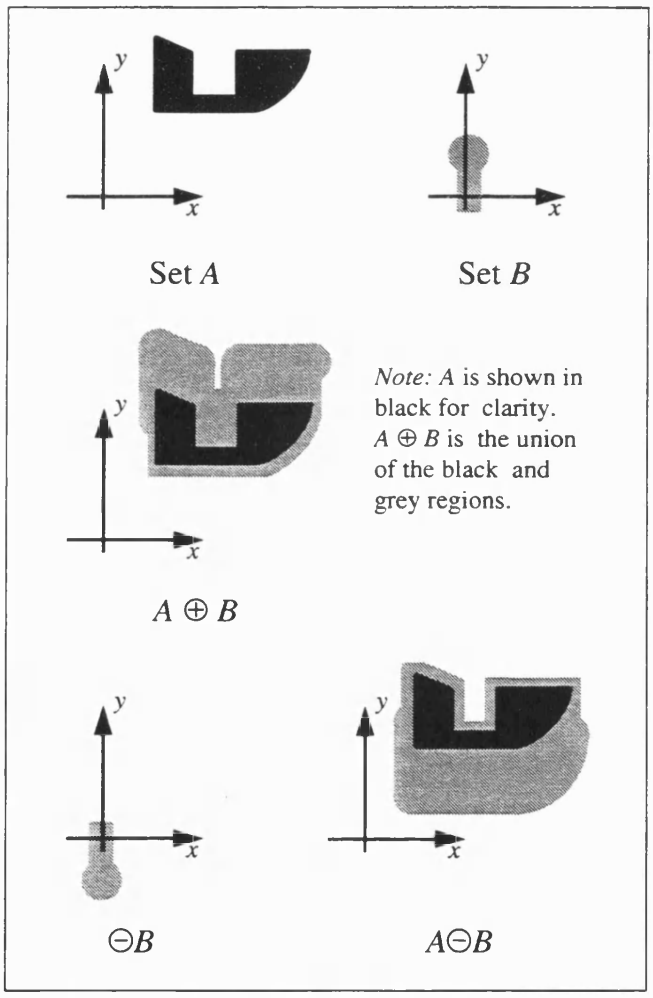


Figure 2.3: **Minkowski operations.**

Lozano-Pérez observed that if B is a nomad which translates and is obstructed by obstacle A, the C-space obstacle to B caused by A is given by $A \ominus B$. Thus a C-space obstacle for a translating nomad can be generated by reflecting the nomad in the origin and swelling the obstacles by the result. This can be achieved for

convex polytopes by reflecting each nomad vertex in the origin and adding the result to each vertex of the obstacle; the C-space obstacle is the convex hull of the resulting point set.

On the subject of an n-dimensional convex polytope, Gouzènes 1984 [35, p. 59] observed the following:

- It is bounded by at least $n+1$ hyperplanes, each defined by $n+1$ coefficients.
- Inclusion requires an operation that is $O(n^3)$.
- The generalisation of 3-D B-rep modelling (the faces-edges-vertices structure) involves $O(n!)$ pointers for a simple convex object.

Gouzenès also points out that a quadratic boundary representation of an n-dimensional C-space obstacle requires $O(n^2)$ coefficients per hypersurface and does not easily support the operations needed for collision avoidance: point membership testing, inclusion and intersection.

Contact-surface

Obstacle growing by Minkowski difference holds for three-dimensional cases and for objects of arbitrary shape, but does not extend intuitively to rotations. However, a precise analytical representation of the C-space obstacles is still achievable since the boundary of any C-space obstacle is a patchwork of surfaces, each of which corresponds to a *contact condition* (Brost 1989 [13])—that is, a specific interaction between an element of the moving object and an element of an obstacle. For example, in the case of a polygonal nomad moving amidst polygonal obstacles, the contact-surface will be a patchwork of two types of nomad-obstacle contact:

Type A contact Where an edge of the nomad makes contact with a vertex of the obstacle (Figure 2.4 (a)).

Type B contact Where a vertex of the nomad makes contact with an edge of the obstacle (Figure 2.4 (b)).

Note that there is not a contact type corresponding to the singularities where vertex-vertex contact occurs—although such configurations are identifiable in a contact surface representation of C-space as the vertices of C-space obstacles (i. e. where two contact surfaces meet).

Figure 2.5 (adapted from Brost 1989 [13]) illustrates how specific contact conditions correspond to faces of a C-space obstacle.

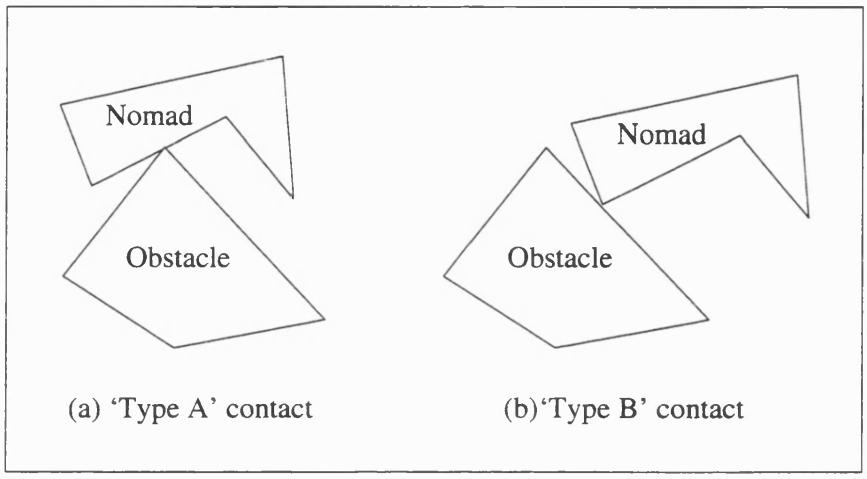


Figure 2.4: **Contact types for polygons.**

Polyhedral cases give rise to three contact types, illustrated in Figure: 2.6:

Type A contact Where a face of the nomad makes contact with a vertex of the obstacle (Figure 2.6 (a)).

Type B contact Where a vertex of the nomad makes contact with a face of the obstacle (Figure 2.6 (b)).

Type C contact Where an edge of the nomad makes contact with an edge of the obstacle (Figure 2.6 (c)).

Note that contacts such as vertex-vertex, vertex-edge, edge-face and face-face are not classified but correspond to edges and vertices of a C-space obstacle—for example, configurations in which a vertex interacts with an edge correspond to edges of the C-space obstacle where a Type C contact face meets a Type A or Type B contact face.

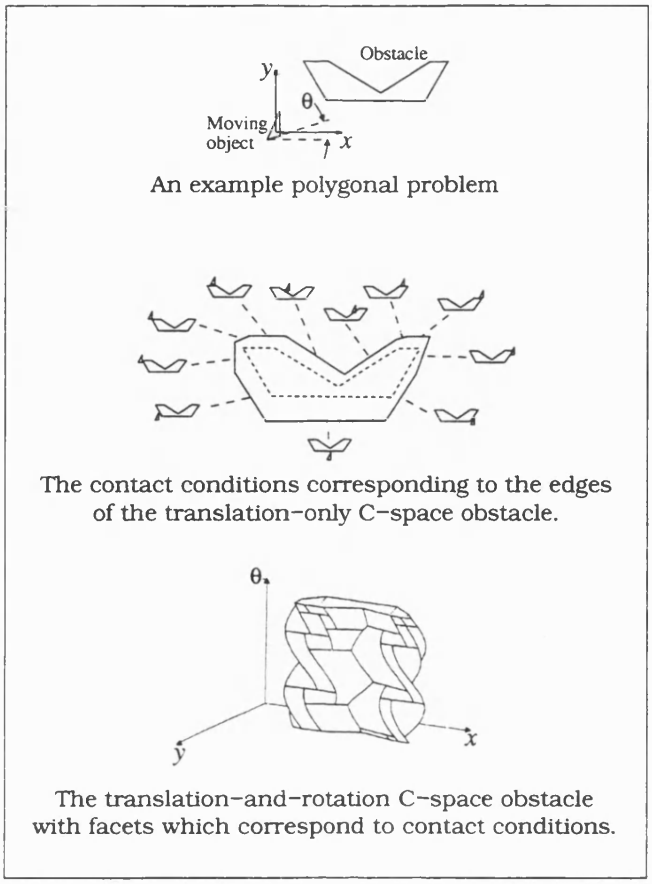


Figure 2.5: **Illustration of contact conditions for a polygonal case after Brost 1989 [13].**

Figure 2.7 (generated using an algorithm described in Chapter 6) illustrates how specific contact conditions correspond to faces of a C-space obstacle for the case of a translating polyhedron. The C-space obstacle to a polyhedron which is free to fully rotate as well as translate has a five-dimensional surface consisting of a patchwork of contact surfaces of type A, B and C.

The key steps in obtaining a configuration space map using a contact-surface approach are:

Formulating the constraint imposed by each contact condition For a nomad to be in contact with an obstacle in a specific contact condition (e. g. for a specific nomad vertex to be in contact with a specific obstacle face) the nomad's configuration is constrained to lie on a surface embedded in the C-space, formulated in terms of all of the degrees of freedom of the

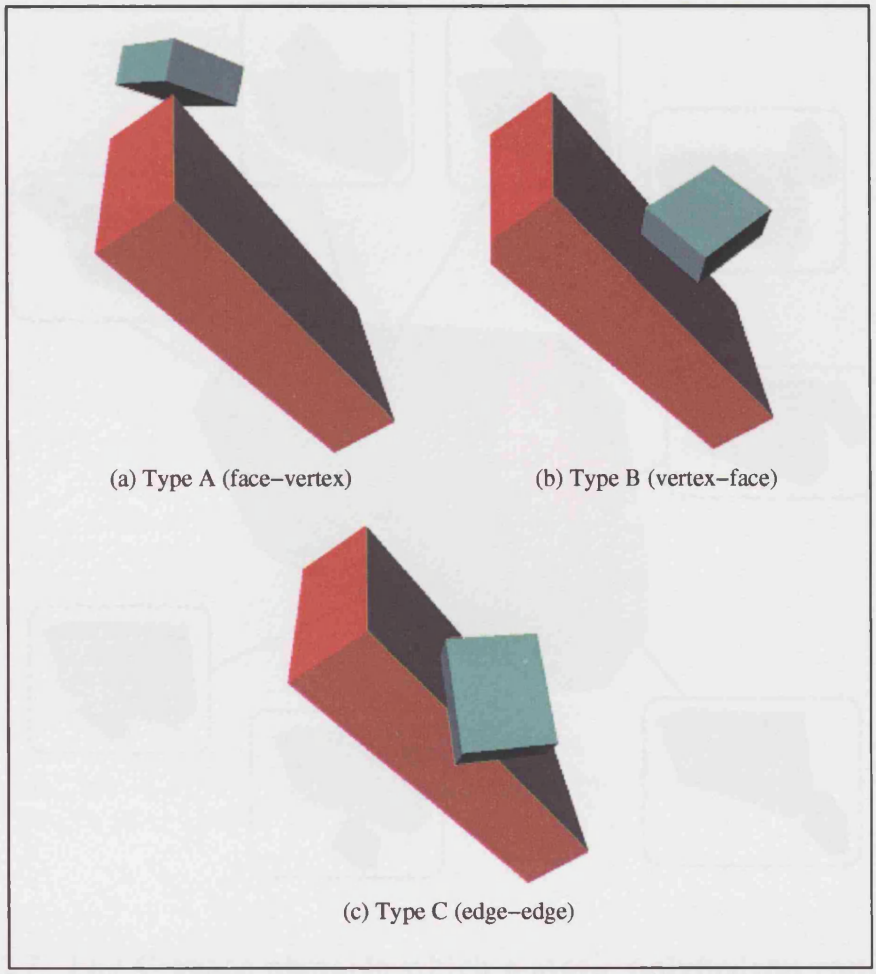


Figure 2.6: **Contact types for polyhedra.**

object. For translation only cases, each of these constraints is a flat sheet in the C-space: for type A and B contact conditions the contact constraint will be parallel to the edge or face involved; for type C contact conditions the constraint surface will be perpendicular to both edges involved.

For polygonal cases where the nomad translates and rotates in the plane, each type A contact condition is a helicoid (Figure 2.8 (a)), and the constraint for each type B contact condition is a sinusoid (Figure 2.8 (b)).

For polyhedral cases which incorporate rotation as well as translation, all constraint surfaces are *ruled surfaces* —slicing any surface at a fixed orientation will result in a flat sheet since it corresponds to a translation-only constraint. Constraint surfaces for type A and type B contacts are multidimensional extensions of the helicoid and sinusoid respectively; type C constraints are more complicated since they are defined as having a surface

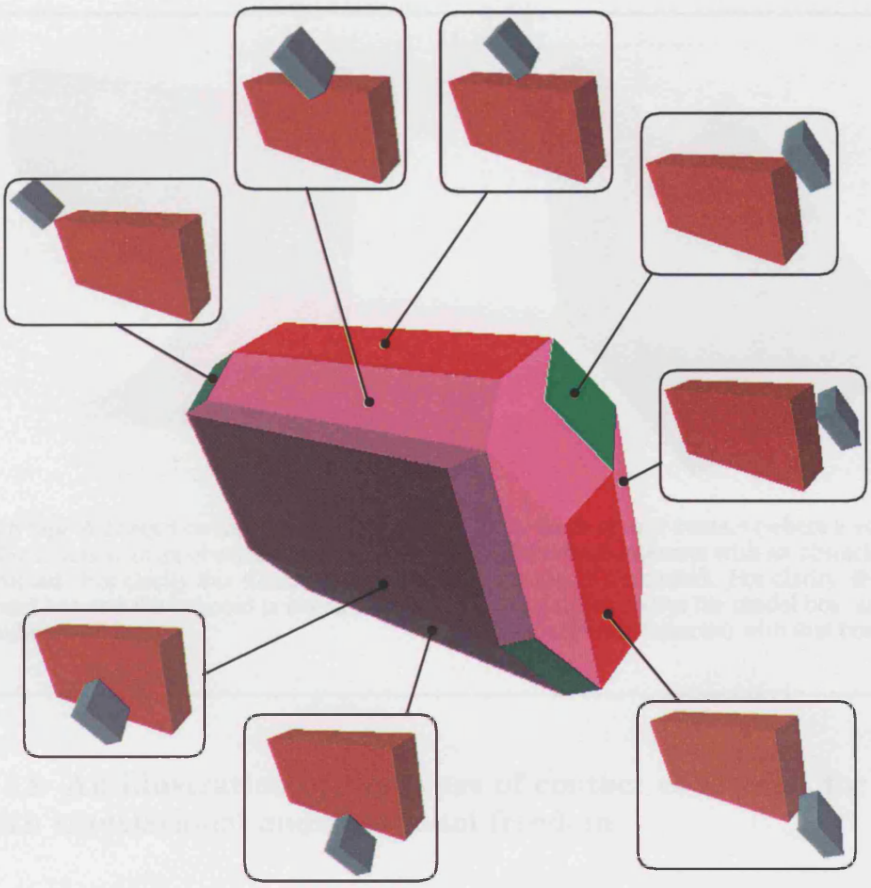
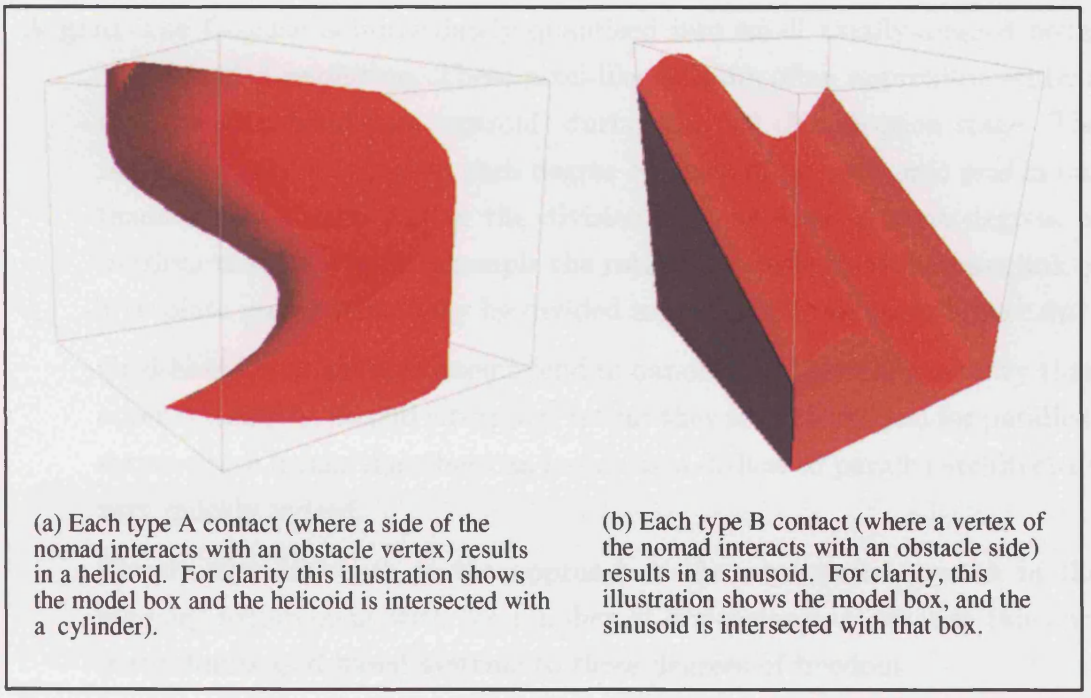


Figure 2.7: **The C-space obstacle which a static polyhedron causes to a polyhedron which can translate in a fixed orientation is a patchwork of contact surfaces of Type A (green), Type B (red) and Type C (pink). The snapshots around the perimeter show example configurations corresponding to some of the contact patches.**

normal perpendicular to both edges which cause the constraint, and the nomad edge rotates according to the orientation parameters. For details of my implementation of a contact-surface representation, see Chapter 6.

Identifying which side corresponds toprohibited configurations This is non-trivial for type C conditions since the edges which are constrained to be in contact have no solid side.

Formulating an applicability condition for each contact condition The contact constraint corresponding to a specific contact condition is only *applicable* for a certain range of orientations—outside of that range the contact cannot occur without the nomad and obstacle interpenetrating. If the nomad is translating in a fixed orientation, each contact constraint is either



(a) Each type A contact (where a side of the nomad interacts with an obstacle vertex) results in a helicoid. For clarity this illustration shows the model box and the helicoid is intersected with a cylinder).

(b) Each type B contact (where a vertex of the nomad interacts with an obstacle side) results in a sinusoid. For clarity, this illustration shows the model box, and the sinusoid is intersected with that box.

Figure 2.8: **An illustration of the types of contact constraint for a polygon with translational and rotational freedom**

applicable or it is not. If the nomad can rotate, each contact constraint has an *applicability condition* associated with it, formulated in terms of the orientation parameters (i.e. the rotational dimensions of the C-space). A contact condition is only applicable in orientations in which the applicability condition is met—this bounds the patch in the rotation dimensions.

2.3.2 Division and classification

An alternative to precise analytical methods is to discretise the C-space into a number of cells and then to use some test to classify each one as safe, prohibited or contact.

Division strategies

The literature describes a number of different division strategies:

A grid The C-space is immediately quantised into small axially-aligned boxes of a specified resolution. These pixel-like cells are often approximated by a configuration point (the centroid) during the cell classification stage. The resolution may be equal in each degree of freedom (an *isotropic grid* in our taxonomy of Figure 2.2) or the division may be finer in some degrees of freedom than others; for example the rotational freedom of the base link of a revolute manipulator may be divided more finely than those higher up.

Grid-based mapmaking systems tend to handle more general geometry than other systems. A second advantage is that they tend to be ideal for parallelisation which means that they can be run on a dedicated parallel architecture very quickly indeed.

Clearly, the drawback to the approach is the exponential growth in the memory requirement with the number of dimensions; in practise this currently limits grid-based systems to three degrees of freedom.

Rasters A one-dimensional C-space map may be represented by a ray that is divided into safe and prohibited segments. A two-dimensional C-space map, instead of being chopped into pixels, can be approximated by a stack of *rasters* (rays with a finite thickness) divided in the same way. A three-dimensional C-space map can be built by stacking resolution-thick 2-D maps, and so on. This approach, which was demonstrated by Lozano-Pérez 1987 [64], increases the smallest cell from being an approximation to a (0-D) point to being an approximation to a (1-D) line. As discussed in Section 2.5.3, rasters are especially appropriate for mapping revolute manipulators. An example of this by Red and Truong-Cao 1985 [80] was illustrated in Figure 1.2.

Axially-aligned boxes An n-dimensional configuration space can be recursively chopped into axially aligned boxes and a classification test can be performed on each box as a whole. In some schemes (for example a 2^n tree, which is a quad-tree in two dimensions and an oct-tree in three) once the decision to chop has been made the position of the chopping hyperplanes is fixed. This is referred to as *blind* division (Zhu and Latombe 1991 [111]). Other schemes are *adaptive*—the position of the chopping hyperplanes is determined by the contents of the box. An adaptive division strategy will tend to bound the C-space obstacles tighter with fewer cells—but this improvement must be weighed against the computational expense of working out where to chop. A bin-tree, which divides into two at each recursion regardless of

dimensionality, can be either blind (chopping the longest degree of freedom in half, for example) or adaptive.

As a method of C-space representation, a tree of axially aligned boxes has the following characteristics:

Memory requirement In practical cases the memory requirement for a box tree may be orders of magnitude less than that of a grid of the same resolution since a large region of safe or prohibited C-space may be represented by a single cell.

Isotropy All degrees of freedom are treated equally, unlike methods such as that of Lozano-Pérez 1981 [65] which treat rotations differently from translations.

Known efficient algorithms As Gouzènes observed (1984 [35, p. 59]), a box supports $O(n)$ algorithms for membership, inclusion and intersection. Efficient algorithms are also available to perform these operations (and other boolean operations) on hierarchical trees of boxes.

Refinability The resolution of a C-space map can be increased locally by adding to (not recomputing) the existing representation (Paden et al. 1989 [73]).

Regularity A regular division (such as a 2^n tree) has the advantage that a cell's neighbours can be visited without maintaining an explicit connectivity graph (Faverjon and Tournassoud 1988 [30, p. 100]). This can account for a considerable memory saving.

A clue to relative safety of cells since large safe cells are sufficiently far from the C-space obstacles, a gross motion which remains far from the obstacles can be obtained quickly by searching for a path which traverses only large cells (Faverjon and Tournassoud 1988 [30, p. 100]).

A safety margin The configuration space approach to spatial planning relies on the accuracy of the models of the nomad and the obstacles *and* on the ability of a nomad to trace a trajectory accurately. Clearly there will be errors in the system, so there is an argument that one of the weaknesses of a divided C-space—that it is always a conservative approximation—is useful in practical applications.

Special Some schemes divide the C-space into cells which are not axially aligned. For example see Lozano-Pérez 1981 [65] or Shiller and Gwo 1993 [90].

Cell classification schemes

Ascertaining whether a C-space cell is safe, prohibited or a mixture of both can be achieved in a number of ways:

Swelling the nomad A fast method which can be used as the first step in classifying a C-space cell is to swell the nomad by the maximum distance any part of it can move (which, in the case of a manipulator, is not trivial to determine—see Lozano-Pérez 1987 [64]). This swollen nomad can then be tested for intersection with the obstacles.

Swept volumes A more accurate alternative to swelling the nomad is to calculate the volume it sweeps out as it moves. This technique can be particularly effective in conjunction with the raster-division method for manipulator mapping (as described in Section 2.5.3). This is because, as observed by Gouzènes 1984 [35], the shape of the swept volume of a particular link exercising its full range of motion is constant—only its position is dependent on the joint angles of the previous links. Verwer 1990 [100] uses a swept volume scheme which employs *bubble hierarchies* whereby the objects and swept volumes are approximated by increasingly accurate arrangements of spheres; this is useful since intersection testing between two spheres is so trivial.

Inverse kinematics Warren 1989 [102] and Adolphs and Nafziger 1990 [3] both divide the C-space of a manipulator into a uniform grid and then map discrete configurations of the end effector into that grid via inverse kinematics. Adolphs and Nafziger store the mapping from workspace to joint space in a look-up table so that changes in the environment can be mapped into the C-space quickly.

Bounded Jacobians Avoiding obstacles places a bound on the Jacobians of points on the nomad or manipulator—this observation can be used to classify cells (Faverjon and Tournassoud 1988 [30],Paden et al. 1989 [73]).

Contact conditions In addition to representing the boundary of a C-space obstacle precisely, analysis of contact conditions can be used to classify a C-space cell such as a raster (see Red and Truong-Cao 1985 [80] and Lozano-Pérez 1987 [64]).

Distance calculations A C-space cell can be classified by calculating the distance to contact between the nomad or manipulator and the obstacles when it is in the centroidal configuration (which will be negative if that configuration is prohibited). If the modulus of the distance calculated is greater than half of the length of a diagonal of the cell, then the cell can be classified as safe or prohibited according the sign. One method to calculate this distance is to use the Minkowski difference operation (Siméon 1988 [91]).

Ralli and Hirzinger 1996 [78] employ a different distance technique to map the C-space of a manipulator. As a pre-mapmaking step, the workspace is filled with a grid and wavefront propagation is used to label each cell with the distance from it to the nearest obstacle. A map of the C-space is then plotted into a grid by using forward kinematics to position each of the links and then only using collision detection if the distance value of the midpoint of each link is less than half of the link's length.

Other useful techniques

Several other useful techniques have been developed for use with the divide-and-classify representation:

Classification propagation This is possible when the classification for one cell enables nearby cells to be classified. For example, Siméon 1988 [91] and Laumond et al 1988. [56] both classify a C-space cell by calculating the distance-to-contact for the centroid of that cell: this information is propagated to any other cells which are contained within a ball with radius of the returned distance.

Restructuring the division Sometimes the representation can be improved by a post-division process which merges cells of the same classification. This can be particularly helpful if the original cells are long thin rasters. Systems which do this include Lozano-Pérez 1987 [64], Siméon 1988 [91] and Faverjon and Tournassoud 1988 [30]. Recent research has also demonstrated how smoother paths can be planned through a grid C-space map after Kohonen-map (Ralli and Hirzinger 1997 [79]) based reorganisation has increased the resolution of the free-space (Ralli and Hirzinger 1996, 1997 [78, 79]).

Selected refinement The cost of global path planning for a manipulator with many degrees of freedom is minimised by only refining the C-space map where a path is likely to proceed. This idea was used by Lozano-Pérez 1981 [65] and is fundamental to many of the later planners (Hasegawa and Terasaki 1988 [38], Verwer 1990 [100], Duelen and Willnow 1991 [26], Bellier et al. 1992 [8], Chen and Hwang 1992 [18]).

C-space obstacle labelling Siméon 1988 [91] labels each C-space obstacle cell to indicate which obstacle(s) in the workspace caused it. Then, the C-space map does not need to be recomputed if an obstacle in the workspace is removed—the path planner is simply told to treat as safe all C-space obstacle cells caused only by the removed obstacle.

2.3.3 Hybrid

A third and final approach to representing a global C-space map is to create a *hybrid* representation which combines elements of those described above. As an example of this, during the mapmaking process described by Lozano-Pérez 1987 [64] obstacles are grown using a Minkowski sum, the C-space is recursively divided into a series of rasters, and cells are merged to result in an adaptively-divided axially-aligned-box representation.

2.4 Mapmaking techniques for a nomad in a static environment

Dealing with a single moving nomad (such as a mobile robot) in a static environment is not only the most simple C-space problem, it is the most fundamental—techniques developed here provide the building blocks for all other C-space mapmakers.

2.4.1 Founding work

One of the characteristics of the C-space approach to spatial planning is that it isolates the kinematic constraints caused by the shapes of objects, leaving other constraints to be dealt with later. As a result, a C-space mapping algorithm may ignore the non-holonomic constraints imposed by the steering mechanism of a mobile robot, for example, and treat it as a free-floating body. Moreover, since most mobile robots are limited to moving on a flat floor and the shapes of the obstacles are approximated as having a constant horizontal cross-section, mobile robot problems are commonly modelled as two-dimensional.

A unconstrained two-dimensional object has three degrees-of-freedom, (two translational and one rotational) resulting in a C-space which is the three-dimensional manifold $\Re^2 \times S^1$ (where S^1 is the unit circle) and which may be represented by the Cartesian space $\Re^2 \times \Re/(2\pi Z)$. A configuration is parameterised by (x', y', θ), where $(x', y') \in \Re^2$ and $\theta \in [0, 2\pi)$. An unconstrained three-dimensional object (which might be a robot in space or a component of a mechanism) has three translational and three rotational degrees of freedom, resulting in a six-dimensional C-space (or a seven-dimensional one if quaternions are used to parameterise rotation).

The Minkowski obstacle-growing technique implemented by Lozano-Pérez (1983 [63]) was able to handle both two- and three-dimensional polytopes, but was unable to handle either rotations or more general geometry. Examining how these two limitations have been overcome provides a convenient way of examining C-space mapmaking techniques for a single nomad.

2.4.2 Handling rotations

Early attempts to handle rotations (for example, Lozano-Pérez 1983 [63] and Jarvis 1983 [46]) considered the problem of a polygonal nomad translating and rotating amidst polygonal obstacles, and treated the rotational degree of freedom differently to the others. The range of rotational values θ was divided into k smaller ranges, and for each θ-range (or θ-slice) a polygonal approximation to the area swept out by the nomad as it rotated was calculated (called a *slice-*

projection); the workspace obstacles were then grown by each slice-projection. This resulted in k sets of grown obstacles (called *θ-slices*) which, when stacked together, approximated the three-dimensional C-space map for the problem. This scheme, which is illustrated in Figure 2.9, has the drawback that the non-isotropic map only allows motions which alternate between translations and rotations.

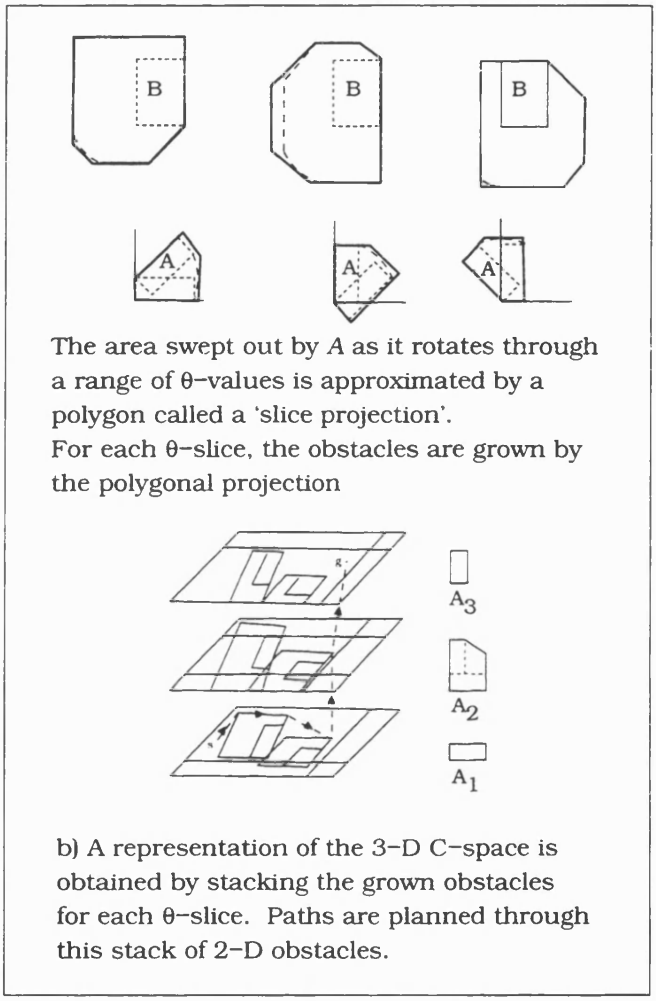


Figure 2.9: **Illustration of the slice projection technique, after Lozano-Pérez 1983 [63].**

The technique of recursive subdivision-and-classification of C-space was introduced by Brooks and Lozano-Pérez 1985 [12], who describe an adaptive division scheme which sits on top of a precise boundary representation calculated using contact conditions. Laumond et al. 1988[56] also demonstrate an adaptive division scheme, this time classifying each cell by calculating the distance to contact. A third adaptive division technique was developed by Zhu and Latombe 1991 [111] which, like that in [12], sits on top of an accurate representation of the C-space

obstacles. Zhu and Latombe introduce *bounded* and *bounding* approximations of C-space obstacles (an approach previously used in computer graphics) and show how they can be used to get a tight decomposition of the C-space obstacles faster than the quad-tree approach.

The majority of C-space mapping research concerned with translating and rotating nomads has concentrated on developing boundary representations of C-space obstacles using the contact condition observation mentioned in Section 2.3.

For example, Donald 1985, 1987 [24, 25] implemented a path planner for a polyhedral object with a full six degrees of freedom based on algebraically representing all contact surfaces, computing their intersections, and defining operators which slide along and jump between one-, two-, three-, four-, five- and six-dimensional surfaces in C-space. Similarly, by formulating the constraints in terms of quaternion parameters and deriving predicates for C-space obstacles as a conjuction (intersection) of constraints, Canny 1988 [15] was able to represent free-space as a semi-algebraic set. Canny then produced a 1-D subset (skeleton) similar to a Voronoi skeleton which captured connectivity and performed path planning using this 'roadmap'. Meanwhile, Avnaim and Boissonnat (on their own 1988 [4] and with Faverjon 1988 [5]) concentrated on the three degrees of freedom of a polygon translating and rotating, developing an efficient and exact path planner (based on the mapping of contact-condition patches) with which they obtained very impressive results. Brost 1989 [13] did similar work and attached *contact information* to each facet which defined the C-space obstacle—an important step towards kinematic analysis of physical interaction. A new approach was introduced by Liu and Onda 1993 [60] who analysed each contact condition for interacting polygons in a local coordinate frame and then plotted the result into a grid representation of the global C-space.

In contrast to all these new analytical methods, Lengyel et al. 1990 [58] attacked the 3-DOF polygon problem using exactly the same method as Lozano-Pérez 1983 [63] but was able to deal with a finer resolution in the rotation dimension up by using parallel dedicated graphics hardware which represented each θ-slice of the C-space obstacles by a rasterised bitmap. A more recent paper by Solano González and Jones 1996 [93] also uses the slice-projection technique to investigate the benefits of parallel computation for C-space mapping.

2.4.3 Handling more general geometry

The first paper to map the C-space of a nomad more complicated than a polygon appears to be that of Laumond 1987 [55], which extended the Minkowski difference obstacle-growing technique from polygons to generalised polygons. Bajaj and Kim also took up the challenge of generalising obstacle-growing but they focused on three-dimensional objects: in [6] (1988) they grow arbitrarily-shaped three-dimensional obstacles by a moving sphere and then in [7] (1990) they grow convex three-dimensional obstacles of arbitrary complexity by an object in that same class. A more recent paper by Kohler and Spreng 1995 [52] also grows convex B-rep objects of (near) arbitrary complexity by each other, only this time back in the two-dimensional domain. Kohler and Spreng's contribution is the development of an efficient obstacle-growing method which, by subdividing the boundary segments of the interacting objects, obtains a precise representation of the grown obstacle without relying on a computer algebra system (unlike the methods of Bajaj and Kim) [52, p. 590].

In 1996 [39], Heegaard suggests that if the objects in the workspace are represented by convex parametric curves, the contact surface can be obtained more efficiently by plotting it in to an alternative kind of C-space. If two interacting planar bodies are represented by parametric curves in terms of parameters ϵ_1 and ϵ_2, Heegaard observes that the contact surface *for both translating and rotating motion* can be easily formulated in terms of ϵ_1, ϵ_2 and d_n, where d_n is the distance between the closest points on the two objects. Note that although the space $\epsilon_1\epsilon_2 d_n$ is not defined in terms of degrees of freedom, it is still a valid configuration space for the specific domain of objects defined by convex parametric curves, since any point within it is sufficient to specify the location of every point in the system.

Finally, at the other end of the C-space representation spectrum, a number of C-space mapping systems have emerged which deal with arbitrary geometry by using a grid division:

- Dehne et al. 1989 [22] use a parallel computer architecture of a type usually used for image processing.
- Kavraki 1993 [48] uses the fast Fourier transform. This technique is based

on the observation that the Minkowski difference of two point-sets which are discretised into two-dimensional arrays is a convolution which can achieved by taking the Fourier transform of the two arrays, multiplying the two transforms pointwise, and then taking the inverse Fourier transform of the result. As a result, the computational cost of this method depends only on the resolution of the grid– the geometry of the objects involved has no effect. Curto and Moreno 1997 [21] have since applied the same technique to map the C-space of a planar revolute manipulator.

- Lin and Chang 1993 [59] create a grid representation of a C-space map using *mathematical morphology* (which is similar but not identical to Minkowski operations) and a *shape decomposition* step which breaks arbitrarily complicated shapes down into simpler elements, as illustrated in Figure 2.10.

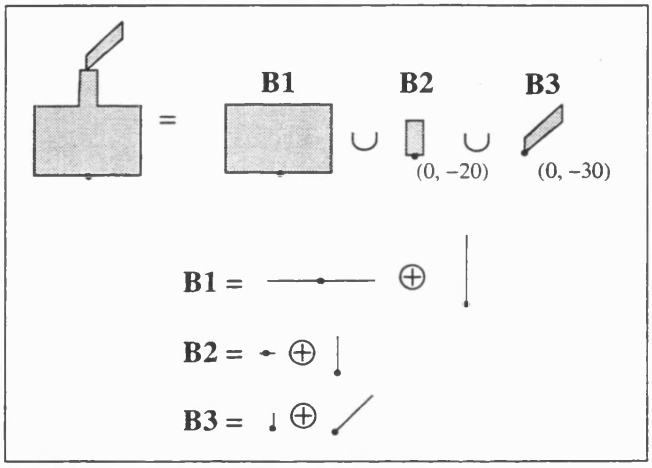


Figure 2.10: **Illustration of the shape decomposition technique of Lin and Chang 1993 [59].**

- Chan et al.1994 [17] attack the problem using neural networks.

2.5 Mapmaking techniques for a single manipulator in a static environment

2.5.1 Founding work

The links of a manipulator are modelled as a series of connected rigid moving objects each of which has one or more degrees of freedom *relative* to the link below. The degrees of freedom of a particular link are specified by the controlled joint—which is usually prismatic or revolute—and the degrees of freedom of the whole manipulator (which define the configuration space) are the sum of the degrees of freedom of the parts. The C-space of a manipulator with n revolute joints and m prismatic ones is the $(n+m)$-dimensional manifold $\Re^m \times (S^1)^n$ (or a subset of that space when the motion of each joint it limited by mechanical stops, Latombe 1993 [53]).

The first path planner of any kind to use the configuration space approach is widely regarded as that of Udupa 1977 [98] which maps the C-space for a manipulator, even though the author did not refer to it in those words[2]. Udupa plans the path of a polar-coordinate manipulator by mapping the problem into a space where the manipulator is reduced to a point, beginning by approximating the links of the manipulator as cylinders and growing the obstacles by their radius. This quick and easy method of reducing the links to line segments has been adopted by many other systems (for example Faverjon 1984 [29], Hasegawa and Terasaki 1988 [38], Warren et al. 1989 [102] and Hwang 1990 [42]). Udupa's work is also notable for its use of two C-space mappings—one of which considers the orientation of the highest link and one of which does not; this idea has since been returned to by others (for example Faverjon 1984 [29] and Hasegawa and Terasaki 1988 [38]).

A second founding work for manipulator C-space mapping is that of Lozano-Pérez 1981 [65] in which he outlines several ideas which have played major roles in C-space mapping research ever since, including hierarchical approximations (described in Section 2.2.2), selective refinement (Section 2.3.2) and the idea of representing a manipulator as a tree structure where each level restricts one

[2]Instead, Udupa used a delightfully Trekesque vocabulary of his own including 'sectoroids', 'pascs' and, my personal favourite, 'pgram motion'.

degree of freedom. Lozano-Pérez goes on to describe an application of the θ-slice scheme outlined in Section 2.4 to the case of a Cartesian manipulator. The system still uses projected swept volumes but introduces a hybrid representation of C-space consisting of a tree where some nodes are axially-aligned boxes and others are arbitrary convex polyhedra.

The rest of this section on C-space mapping for a manipulator is divided according to the representation scheme used for the C-space map.

2.5.2 Analytical representations

The combination of multiple rotational degrees of freedom and the fact that moving a link moves every higher link makes it difficult to represent the boundary of the C-space obstacles for a revolute manipulator. In [32, 33] (1989), however, Ge and McCarthy go some way to achieving this by representing the configuration of the end effector using *dual quaternions* in the *image space.* The set of possible configurations for the end effector is bound by two constraints: the reachability constraint, determined by the links of the manipulator and the contact constraint determined by contact between the effector and the obstacles. Ge and McCarthy exploit the fact that both types of constraint can be represented by algebraic and parametric forms in the image space which can then be intersected easily. When this intersection is parameterised piecewise in terms of the joint angles by using inverse-kinematics, the result is an explicit representation of the C-space obstacles to the end-effector. This method is noteworthy, but the collisions between the links and the obstacles are not considered so the C-space obstacles do not represent a true C-space map for the manipulator.

A true mapmaker which uses an analytical boundary representation is that of Hwang 1990 [42] who approximates a three-dimensional revolute manipulator as linked cylinders, and convex polyhedral obstacles as a collection of planar triangular facets. Hwang reduces the manipulator links to lines (like Udupa 1977 [98]) and calculates the intersection conditions between the first link and each triangular patch, unioning the results to derive boundary equations for the first joint variable. He goes on to show how the boundary equation of each higher joint variable can be solved explicitly in terms of the lower joint variables. Interestingly, Hwang confesses that although a precise representation of the C-space

obstacles can be obtained in this way, the computational expense of intersecting the boundary equations means that any practical path planner must first convert the map into a representation that is easier to work with.

Discretisation of the obstacles is taken one step further by Zhao, Farooq and Bayoumi 1995 [110] and Ma, Li, Yang and Chang 1995 [66] who treat the obstacles as a discrete set of points. Zhao, Farooq and Bayoumi [110] discretise the workspace into a grid and then, for each cell which contains the surface of an obstacle, use a variation on inverse-kinematics to obtain parametric equations for the C-space obstacles caused to the manipulator by a point-obstacle at the cell's centroid; the C-space obstacle for the complete obstacle set is obtained by unioning the results. Ma, Li, Yang and Chang 1995 [66] use a similar approach, but, dealing with obstacles composed of axially-aligned boxes, reduce the amount of contact-surfaces generated by isolating the "fundamental points"—the small number of points on the surface of the obstacle set whose C-space obstacles contribute to the final result.

2.5.3 Division and classification

Most of the divide-and-classify techniques which have been used to generate a C-space map for a manipulator been covered in previous sections. However, one domain-specific technique exploits the tree nature of a manipulator to obtain a raster representation (see, for example Lozano-Pérez 1987 [64]):

Starting at the base joint,

1. Determine what ranges of joint angles (if any) are completely safe from interference regardless of what joint angles the higher links take. One method of achieving this is by calculating the volume swept out by the higher links then checking that volume for intersection with the obstacles.

2. Discretise the complement of those safe ranges into equal-sized intervals of the desired resolution.

3. For each of these ranges of angles, consider the manipulator when the joint is fixed at the centroid value of that interval. Then consider the effect of

the next joint by recursively entering this process at step (1).

2.5.4 Hybrid

Shiller and Gwo 1993 [90] present a mapmaking system which uses a hybrid representation. The authors establish an analytical representation of two-dimensional C-space obstacles, then divide the C-space using an adaptive division strategy. The safe leaf cells which result are bounded by axially-aligned lines on some sides but the precise C-space obstacle surface on the others (as illustrated in Figure 2.11). Significantly, the entire boundaries of the C-space obstacles are not evaluated—the only evaluations are of intersections and tangency points with axially aligned rays.

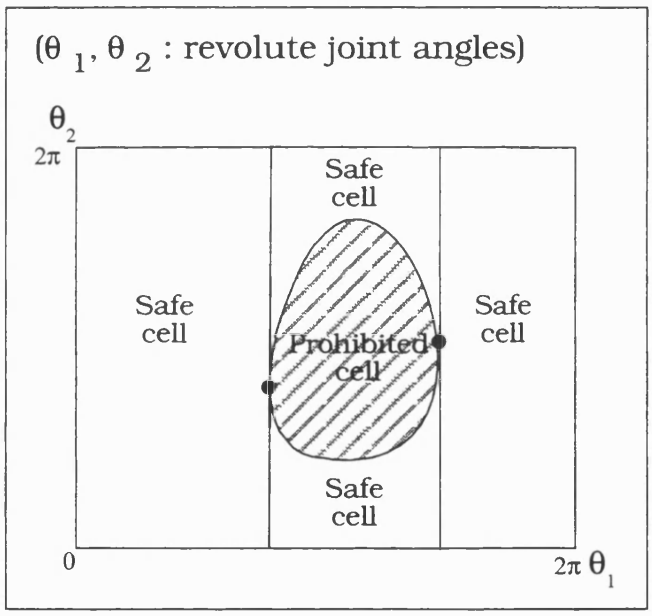


Figure 2.11: **An illustration of the hybrid division scheme of Shiller and Gwo 1993 [90].**

2.5.5 Related techniques

Finally in this section, a brief mention of some techniques that have emerged for non-exhaustive-mapping which may be applicable to the global C-space mapping problem:

'Buck passing' The SANDROS planner described by Chen and Hwang 1992 [18] plans a path by mapping the C-space only until a more efficient local planner can handle the situation adequately. The result is a system which the authors claim gives a performance time proportional to task difficulty.

Topology analysis Maciejewski and Fox 1993 [67] borrow techniques from the analysis of kinematically redundant manipulators to extract information about the *topology* of the configuration space obstacles for a three-DOF manipulator.

Contour tracing Tso and Liu 1993 [97] map the boundary of a two-dimensional C-space obstacle by finding a configuration on the surface and then using contour information to trace the boundary around the perimeter of the obstacle.

2.6 Classification tables

Table 2.1 explains the abbreviations used in the following tabular summary of the C-space mapmaking literature:

Table 2.2 C-space mapmakers for a single nomad in a static environment.

Table 2.3 C-space mapmakers for a manipulator in a static environment.

2.7 Summary & conclusions

In summary, the key decisions to be made regarding how the nomad and its environment are modelled are:

What limits are placed on the shapes of the object models Two-dimensional problems are much easier to handle than three-dimensional ones. Polytopes are easier to handle than generalised polytopes, which are easier than more

Criteria	*Abbreviation*	*Meaning*
Any	(...)	The researcher(s) suggest that the method could be applied to the bracketed case, but details of the application are not given—eg. 2D(3D) indicates that the paper states that 3-D problems can be solved, but only details a 2-D solution.
Degrees of freedom	(n)	The researcher(s) suggest that the method could be extended to an arbitrary number of degrees of freedom.
C-space degrees of freedoms	PM	Only a *partial map* is made
Allowable geometry	conv- cyl poly gen-poly arb	convex cylindrical polytopes generalised polytopes more general than generalised polytopes
Approximation strategy (see Section 2.2.3)	P-P P-A A-P A-A	Precise models, precise C-space mapping Precise models, approximate C-space mapping Approximate models, precise C-space mapping Approximate models, approximate C-space mapping
C-space representation	grown-obs	A B-rep of the translation-only C-space is used.
	c-surface	A B-rep of a C-space containing rotations is used.
	adapt-div	An adaptive division strategy is used.
	grid	A uniform grid division is used.
	raster	A raster representation is used.
	rep1/*rep2*	Both *rep1* and *rep2* are used.

Table 2.1: **Abbreviations used in Tables 2.2** and 2.3.

Paper	*Dimens-ionality*	*Rotations allowed?*	*Allowable geometry*		*Approx. strategy*	*C-space rep.*
			Nomad	*Obstacle*		
Lozano-Pèrez and Wesley, 1979 [62]	2D(3D)	yes for 2D	conv-poly	conv-poly	A-A	grown-obs /grid
Jarvis, 1983 [46]	2D(3D)	yes for 2D	poly	conv-poly	A-A	grown-obs /grid
Lozano-Pèrez, 1983 [63]	2D(3D)	yes for 2D	poly	poly	A-A	grown-obs /grid
Gouzenés, 1984 [35]	2D(3D)	yes	poly (arb)	poly (arb)	A-A	adapt-div /raster
Brooks and Lozano-Pérez, 1985 [12]	2D	yes	poly	poly	A-A	adapt-div /c-surface
Donald, 1985 [24]	3D	yes	poly	poly	A-P	c-surface
Laumond, 1987 [55]	(2D)	(no)	(gen-poly)	(gen-poly)	A-P	(grown-obs)
Avnaim, Boissonnat and Faverjon, 1988 [5]	2D	yes	poly	poly	A-P	c-surface
Avnaim and Boissonnat, 1988 [4]	2D(3D)	yes for 2D	poly	poly	A-P	c-surface
Bajaj and Kim, 1988 [6]	3D	no	sphere	arb	A-P	grown-obs
Laumond et al., 1988 [56]	2D	yes	poly	poly	A-A	adapt-div
Brost, 1989 [13]	2D	yes	poly	poly	A-P	c-surface
Dehne, Hassenklover and Sack, 1989 [22]	2D	no	conv-arb	arb	A-P	grid
Paden, Mees and Fisher, 1989 [73]	(2D)	(no)	(gen-poly)	(gen-poly)	A-A	(2^n-tree)
Bajaj and Kim, 1990 [7]	3D	no	conv-arb	conv-arb	P-P	grown-obs
Lengyel et al., 1990 [58]	$2\frac{1}{2}$D (3D)	about 1 axis	poly	poly	A-A	grown-obs /strips/grid
Verwer, 1990 [100]	(3D)	(yes)	(poly)	(poly)	(A-A)	(2^n-tree)

Table 2.2: **C-space mapmakers for a single nomad in a static environment (*cont.*).**

Paper	*Dimens-ionality*	*Rotations allowed?*	*Allowable geometry*		*Approx. strategy*	*C-space rep.*
			Nomad	*Obstacle*		
Zhu and Latombe, 1991 [111]	2D	yes	poly	poly	A-A	adapt-div /c-surface
Halperin, Overmars and Sharir, 1992 [36]	2D	yes	L-shape	poly	A-P	c-surface
Kavraki, 1993 [48]	2D (3D)	yes for 2D	arb	arb	A-P	grid
Lin and Chang, 1993 [59]	2D (3D)	no	arb	arb	A-P	grid
Liu and Onda, 1993 [60]	2D (3D)	yes for 2D	poly	poly	A-A	grid
Chan, Tam and Leung, 1994 [17]	2D	yes	conv-arb	arb	A-A	grid
Kohler and Spreng, 1995 [52]	2D	no	conv-arb	conv-arb	P-P	grown-obs
Heegaard, 1996 [39]	2D	yes	conv-arb	conv-arb	P-P	c-surface
Wise and Bowyer, 1996 [105]	2D(3D)	no (yes)	gen-poly (arb)	gen-poly (arb)	A-A	bin-tree (/c-surface)
Curto and Moreno, 1997 [21]	2D	yes	arb	arb	A-P	grid

Table 2.2: (***cont.***) **C-space mapmakers for a single nomad in a static environment.**

Paper	*Dimensionality*	*DOFs*		*Allowable geometry*		*Approx. Strategy*	*CS-rep*
		Nomad	*CS-map*	*Nomad*	*Obstacle*		
Udupa, 1977 [98]	2D (3D)	3	3	cyl	poly	A-A	grown-obs
Lozano-Pérez, 1981 [65]	3D	4(7)	4(7)PM	conv-poly	conv-poly	A-A	grown-obs /adapt-div
Faverjon, 1984 [29]	3D	6	3	conv-gen-poly	conv-gen-poly	A-A	grown-obs /grid /2^n-tree
Gouzenés, 1984 [35]	2D (3D)	3	3	poly (arb)	poly (arb)	A-A	adapt-div /raster
Laugier and Germain, 1985 [54]	3D	6	4	conv-poly	conv-poly	A-A	adapt-div
Red and Truong-Cao, 1985 [80]	2D	2	2	poly	poly	A-A	raster
Lozano-Pérez, 1987 [64]	3D	6 (n)	4 (n)	conv-poly	poly	A-A	grown-obs /raster /adapt-div
Faverjon and Tournassoud, 1988 [30]	3D	6	3	gen-poly (arb)	gen-poly (arb)	A-A	raster /2^n-tree
Hasegawa and Terasaki, 1988 [38]	3D	6	3	gen-poly	gen-poly	A-A	grid
Siméon, 1988 [91]	3D	4	4	poly	poly	A-A	raster /adapt-div
Paden, Mees and Fisher, 1989 [73]	2D	2	2	gen-poly	gen-poly	A-A	2^n-tree
Warren, Danos and Mooring, 1989 [102]	3D	2	2	cyl	conv-poly	A-A	grown-obs /grid
Adophs and Nafziger, 1990 [3]	3D	6	3	poly?	poly?	A-A	grid
Branicky and Newman, 1990 [11]	3D	3	3	conv-poly	poly	A-A	grid

Table 2.3: **C-space map-makers for a manipulator in a static environment (*cont.*).**

Paper	*Dimens-ionality*	*DOFs*		*Allowable geometry*		*Approx. Strategy*	*CS-rep*
		Nomad	*CS-map*	*Nomad*	*Obstacle*		
Ge and McCarthy, 1990 [33]	3D	6	6 PM	conv-poly	conv-poly	A-P	c-surface
Hwang, 1990 [42]	3D	2 (n)	2 (n)	cyl	poly	A-A	c-surface
Verwer, 1990 [100]	3D	5	5 PM	poly	poly	A-A	2^n-tree
Newman and Branicky, 1991 [72]	3D	3	3	gen-poly	gen-poly	A-A	grid
Bellier et al., 1992 [8]	3D	6	3	arb	arb	P-A	2^n-tree
Chen and Hwang, 1992 [18]	3D	6	6 PM	poly	poly	A-A	grid /adapt-div
De Pedro and Rosa, 1992 [23]	2D	2	2	line	points	A-A	grid
Shiller and Gwo, 1993 [90]	3D	2 (n)	2 (n)	poly	poly	A-P	c-surface /adapt-div
Tso and Liu, 1993 [97]	3D	3	3 PM	conv-poly	conv-poly	A-P	c-surface
Ma et al., 1995 [66]	3D	6 (n)	6 (n)	cyl	axially-aligned boxes	A-P	c-surface
Zhao, Farooq and Bayoumi, 1995 [110]	3D	3 (n)	3 (n)	cyl	poly	A-P	c-surface
Ralli and Hirzinger, 1996 [78]	3D	≤ 6	≤ 6	cyl	arb	A-A	grid (with Kohonen reorganisation)
Duelen and Willnow, 1991 [26]	3D	5 (n)	5 (n) PM	gen-poly	gen-poly	A-A	2^n-tree

Table 2.3: (***cont.***) **C-space map-makers for a manipulator in a static environment.**

arbitrary geometry. Convex objects are much easier to handle than non-convex objects—but the distributive nature of the C-space mapping makes it possible to handle non-convex problems if the objects can be broken into convex components.

Which representation scheme to use The most common representation schemes for the nomad and environment are B-rep and pixel/voxel decomposition. B-rep is more flexible since it can be used in both analytical and divide-and-classify C-space mapping algorithms, but the choice will depend on the available data—for example, a discrete representation may be available directly from sensors.

Whether to introduce approximation The computational expense of C-space mapping may be eased by introducing approximation at the object modelling stage and/or the C-space mapping stage.

The most central decision regarding the C-space mapmaking algorithm is which representation scheme to use for the map itself. The most common options are:

Analytical representation A precise analytical representation of a C-space obstacle is achievable by performing Minkowski operations if the nomad is a translating object, or otherwise by calculating patches of the contact surface corresponding to specific contact conditions. Such representations tend to be both compact and fast to compute. However, operations such as membership testing, inclusion and boolean operations may be expensive. A greater restriction is that precise solutions have only been implemented for arbitrary-shaped objects for the case of a single object translating in a fixed orientation, although they are available for unconstrained objects and manipulators if the objects are modelled as polytopes.

Divide-and-classify The approach of dividing the C-space into discrete cells and classifying each as safe, prohibited or contact is applicable to a wide range of problems and there are a host of different division strategies, cell-classification schemes and other useful techniques. Discretising the C-space into a grid enables algorithms such as the fast Fourier Transform to be used which are independent of the shape of the object represented, but the memory requirement grows exponentially with dimensionality of C-space.

Hybrid schemes A common approach is to develop a hybrid representation. For example, some mapmakers use a boundary representation in the translation dimensions but discretise the rotation dimensions.

The state of the art of C-space mapping for a single nomad is that a precise C-space map can be obtained for precise object models for both two- and three-dimensional cases if the nomad translates in a fixed orientation and can be broken into convex pieces (Bajaj and Kim 1990 [7]). Rotations make a boundary representation much more difficult, but a precise C-space map can be made for the two-dimensional case as long as the objects are approximated by polytopes (for example Avnaim and Boissonnat 1988 [4]) or as a collection of discrete cells (for example Kavraki 1993 [48]). Six-dimensional C-space obstacles are so difficult to represent that only one paper (Donald 1985 [24]) obtains a global C-space map for a three-dimensional floating object with full degrees of freedom, and that map only enables paths to be planned which slide along on the contact-surface.

C-space mapmaking for manipulators has not been concerned with accurate object models since in practical applications the manipulator should not get close enough to the obstacles for precise geometry to play a role—instead the focus has been on increasing the dimensionality of the C-space maps. This has mainly been achieved by using increasingly intelligent adaptive divide-and-classify techniques and selective refinement of the map, culminating in C-space maps of up to six dimensions (for example Chen and Hwang 1992 [18]). Precise C-space maps have also been obtained for manipulators by representing the links as cylinders and treating obstacles as a collection of facets (Hwang 1990 [42]) or points (Ma et al. 1995 [66], Zhao, Farooq and Bayoumi 1995 [110])—this has also enabled six dimensional C-space maps.

In conclusion, the literature demonstrates that C-space mapmaking is an ongoing area of research, driven by the wide range of applications and the challenge of increasing the dimensionality of the maps and range of allowable geometry taken as input. One significant goal is to produce precise C-space maps of six dimensions or more which support efficient operations for membership testing, ray-tracing and path-planning through safe as well as contact regions.

Chapter 3

Set-theoretic modelling at Bath

3.1 Introduction

This chapter provides important background to this thesis: it gives an overview of the way set-theoretic modelling is implemented by the group at the University of Bath (of which I was a member) and shows the evolution of the multidimensional geometric modeller used to implement the algorithms described in later chapters.

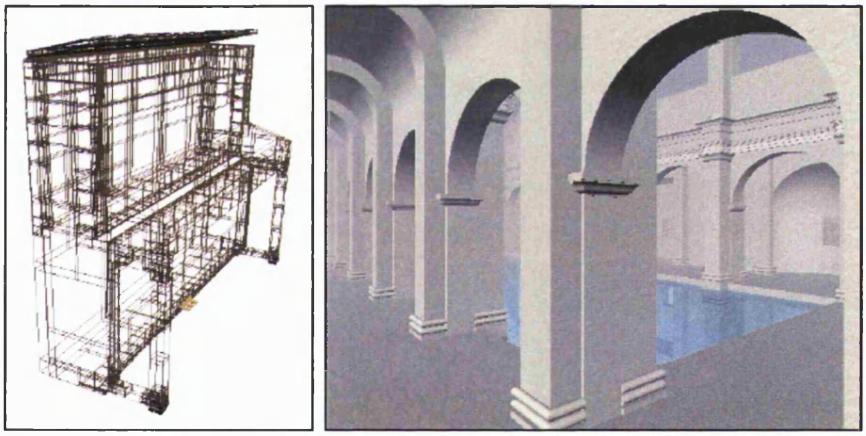
Three set-theoretic geometric modelling kernels are described, each of of which is a library of classes and procedures implemented in C++: svLis, the three-dimensional modeller which was the inspiration for the other two modellers; hypersvLis a pseudo nine-dimensional extension to svLis which was developed for an initial investigation into the omnimodel approach described in Chapter 5; and svLis-m, the multidimensional modeller I helped develop during the course of this research. The modellers are compared and contrasted in terms of the motivation behind them and their use of the following: dimensions, points and boxes, linear halfspaces, other primitives, sets, models, interval arithmetic, and pruning and recursive division.

3.2 svLis

3.2.1 Motivation

SvLis (Bowyer [10]) is a three-dimensional geometric modelling kernel which uses a purely set-theoretic representation. It was developed by Adrian Bowyer (with contributions from members of the Bath Geometric Modelling group) as a tool with which to perform research into set-theoretic modelling techniques, and to provide a set-theoretic kernel which higher-level systems could use to store their geometry (all current commercial CAD systems rely on B-rep kernels). SvLis is distributed by the University of Bath and Information Geometers Ltd; the latest version is available free from: `http://www.bath.ac.uk/~ensab/G_mod/Svlis/svlis.html`.

Since svLis itself is not intended as a CAD system it does not supply a sophisticated user interface for building and editing objects. However, a model can be viewed—either as a series of facets in a polygon display program (Figure 3.1 (a)) or as a ray-traced image (Figure 3.1 (b)).



(a) A wireframe view of a piano (modelled in svLis by David.Eisenthal)

(b) A ray–traced image of the Great Bath, Bath (modelled in svLis by David Lavender)

Figure 3.1: **Images of svLis models.**

3.2.2 Dimensions

All svLis objects are explicitly placed in a three-dimensional Euclidean space, $\{xyz\}$. Since svLis is a *geometric* modeller as opposed to merely a solid modeller, objects of zero thickness such as wires and sheets can be represented as well as solids. Boxes which have zero thickness in one or more dimensions can be constructed and have sets placed in them, but two-dimensional models are not fully supported.

3.2.3 Points and boxes

Points and vectors are implemented using a class which explicitly stores x, y and z coefficients; axially-aligned boxes are defined by explicit intervals in the x, y and z dimensions.

3.2.4 Linear halfspaces

In svLis, planes are defined in the implicit form:

$$Ax + By + Cz + D = 0 (A, B, C, D \ const.)$$

This is stored as a vector $(A, B, C)^T$, which is the plane's normal, and a real, D, which is the *offset* (the smallest distance between the plane and the origin). Thus a planar halfspace is represented by the form:

$$Ax + By + Cz + D \leq 0$$

The meaning of the coefficients are illustrated in Figure 3.2 which shows the two-dimensional equivalent.

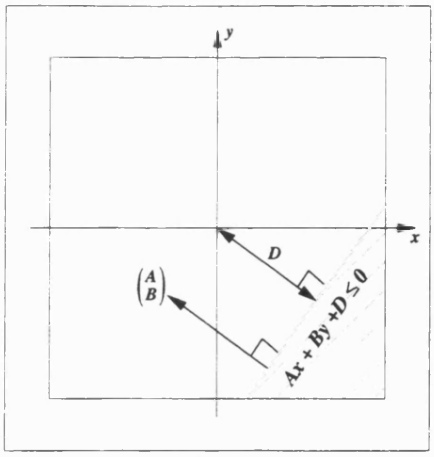


Figure 3.2: **The implicit representation of a halfspace**

Note the following properties of this representation:

- The vector $(A, B, C)^T$ is the plane's normal, pointing from the SOLID region into AIR.
- For any point $p = (p_x, p_y, p_z)$, the *signed distance*, d, from p to linear halfspace H (with normal $H_{normal} = (H_A, H_B, H_C)^T$ and offset H_D) is given by:

$$d = H_A.p_x + H_B.p_y + H_C.p_z + H_D \equiv p.H_{normal} + H_D \qquad (3.1)$$

- The magnitude of d is the Euclidean distance between the point and the surface. Even more significantly, the sign of d indicates which region of space the point is in relative to the halfspace—a negative value indicates that the point is in the SOLID region, a positive value indicates AIR, and zero indicates the surface.

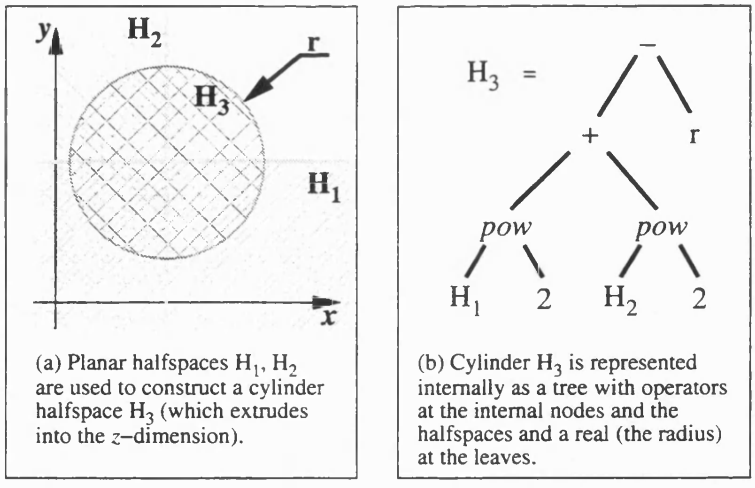
  This sign convention provides the representation of all solids in svLis, hypersvLis and svLis-m.

- Adding or subtracting a constant to a halfspace has no effect on the surface normal, but changes the value of D, which corresponds to the signed distance of origin with respect to the halfspace. Thus subtracting a constant E from a halfspace will offset the surface by a distance $|E|$; if E is positive (resp. negative) the solid region is increased (resp. decreased).

3.2.5 Other primitives

Planar halfspaces are not only the most simple primitive—they are the building blocks for all others, which are built by combining planar halfspaces and reals using arithmetic and transcendental operators—svLis supports primitive addition, subtraction, division (by a real), multiplication, exponentiation (to a positive integer power), sine, cosine and signed square root (that is, the square root of the absolute value of the primitive, with the original sign re-attached). This method of representation, which I refer to as the *linear-halfspace-basis* (a generalisation of the planar basis) can be used to construct any object with an implicitly defined surface—svLis supplies five standard primitives (cylinder, sphere, cone, torus and cyclide) and users may build any other they wish.

Within svLis, primitive are stored as a tree with operators at the internal nodes and halfspaces and reals as the leaves (Figure 3.3). It is worth noting that the implementation of this structure (and the equivalent for sets) is strictly a graph since a reference scheme is used whereby many of the nodes are simply a pointer to another node—although this is an implementation detail it is this scheme which makes the approach practicable.



(a) Planar halfspaces H_1, H_2 are used to construct a cylinder halfspace H_3 (which extrudes into the z–dimension).

(b) Cylinder H_3 is represented internally as a tree with operators at the internal nodes and the halfspaces and a real (the radius) at the leaves.

Figure 3.3: **A cylinder primitive is represented as a tree.**

For any primitive, a potential value can be obtained for a specific point by computing the signed distance of that point relative to each planar halfspace at the leaves, then combining the results according to the operators in the primitive (for example, the potential value for an addition node is the sum of the values

for the two children). As with a single planar halfspace, the sign of the result indicates which region of space the point is in with regard to the primitive, with the convention that negative values indicate SOLID. Thus, testing which region a point is in with respect to a primitive (called a *point membership test* operation) is achieved by computing the potential value of the point with respect to the primitive, and examining the sign of the result. Note, however, that in the general case the magnitude of the potential value is not the Euclidean distance to the surface since the arithmetic and transcendental operators distort the field.

In addition to enquiries about the potential value of a primitive, enquiries can be made about the *gradient* of the potential field. The x, y and z components of the gradient vector are each themselves a primitive, into which a point or box can be fed to obtain the appropriate value or range. Enquiries about the range of gradients within a box enable facetted images to be produced with appropriate shading (see Figure 3.1 (a)).

3.2.6 'Thin' primitives

As pointed out earlier, svLis and its relatives are *geometric* modellers as opposed to *solid* modellers since they are able to represent non-manifold objects such as sheets of zero-thickness. These are obtained by applying the 'abs' operator to a primitive—which makes the potential field of the primitive return the absolute value of what it was previously. This forces the potential field to be positive everywhere except the surface, thus making a 'thin' set. These have a number of applications, particularly when geometric constraints are modelled (see Eisenthal [27]).

3.2.7 Sets

Sets are built by combining primitives with the boolean operators union ($\cup$), intersection ($\cap$) and complement(!). Difference and symmetric difference are also available to the user, but these are internally represented as combinations of the first three. Sets are stored as a tree with operators as the internal nodes and primitives at the leaves.

A point membership test against a set can be achieved by testing the point against each primitive in the tree and combining the results according to the operators, where the union operator means the minimum value from the two children is taken, intersection means take the maximum value, and complement means negate the value of the child.

In general it is not necessary to test the point against every primitive in the tree since any set intersected with AIR is AIR so as soon as a child of an intersection operator is found to have an positive (AIR) value, the point is known to be AIR with respect to the whole set tree. Similarly, any set unioned with SOLID is SOLID. To take maximum advantage of this, svLis arranges the tree such that the most simple branches (which are most likely to evaluate to SOLID or AIR) are evaluated first.

3.2.8 Models

A svLis model is a list of sets embedded in an axially aligned box. The box locates the set in space, defining a region of interest. Primitives which lie entirely outside of the box are removed from the set tree using a process called *pruning* which exploits interval arithmetic.

3.2.9 Interval arithmetic

Arithmetic operators are well defined for intervals (for a treatise see Moore [69]), and appropriate interval equivalents can be defined for each of the transcendental operators supported by svLis; maximum and minimum are also well defined for intervals. Consequently, a range of potential values can be obtained for an axially-aligned box in the same way a single value is obtained for a point: The x, y, and z intervals are placed into Equation 3.1 for each of the planar halfspaces at the leaves of the primitives, and the ranges of values are combined using the rules described above until a range is obtained representing all the values which points in the box have with respect to the set.

Note that in general interval arithmetic may cause the result interval to grow

larger than it should. If a set contains only planar halfspaces or quadric surfaces arranged in canonical form (Voiculescu [101, p. 22]), the answer will be precise, but in other cases the range obtained for a primitive may grow to include zero—indicating that the surface of the primitive passes through the box—when in reality it does not. However, interval arithmetic is always *conservative*—although it might erroneously indicate surface, if it indicates SOLID or AIR then the surface is guaranteed not to pass through the box.

Current algorithms systematically reduce inaccuracies by recursively dividing the box (see below) which is relatively computationally cheap due to the efficiency of the algorithms. Meanwhile, research into methods to improve the accuracy of interval arithmetic, along with alternatives, are under way (Voiculescu [101]).

3.2.10 Pruning & recursive division

Note the following identities:

- *anything* ∩ AIR is AIR
- *anything* ∪ SOLID is SOLID
- *anything* ∪ AIR is itself
- *anything* ∩ SOLID is itself

Armed with these and interval arithmetic, it is possible to (conservatively) remove from the set tree every primitive whose surface lies completely outside of the model's box. Depending on the operator above it, each set which has a potential range which is all negative or all positive is either removed from the tree, or enables the whole tree to be reduced to SOLID or AIR respectively. Only primitives which have a potential value which straddles zero (indicating SURFACE) remain in the set tree.

This process, called *pruning the set to the box* is illustrated in Figure 3.4.

Pruning enables a complicated model to be systematically broken into more simple pieces by recursive division. Typically this takes the form of binary spatial

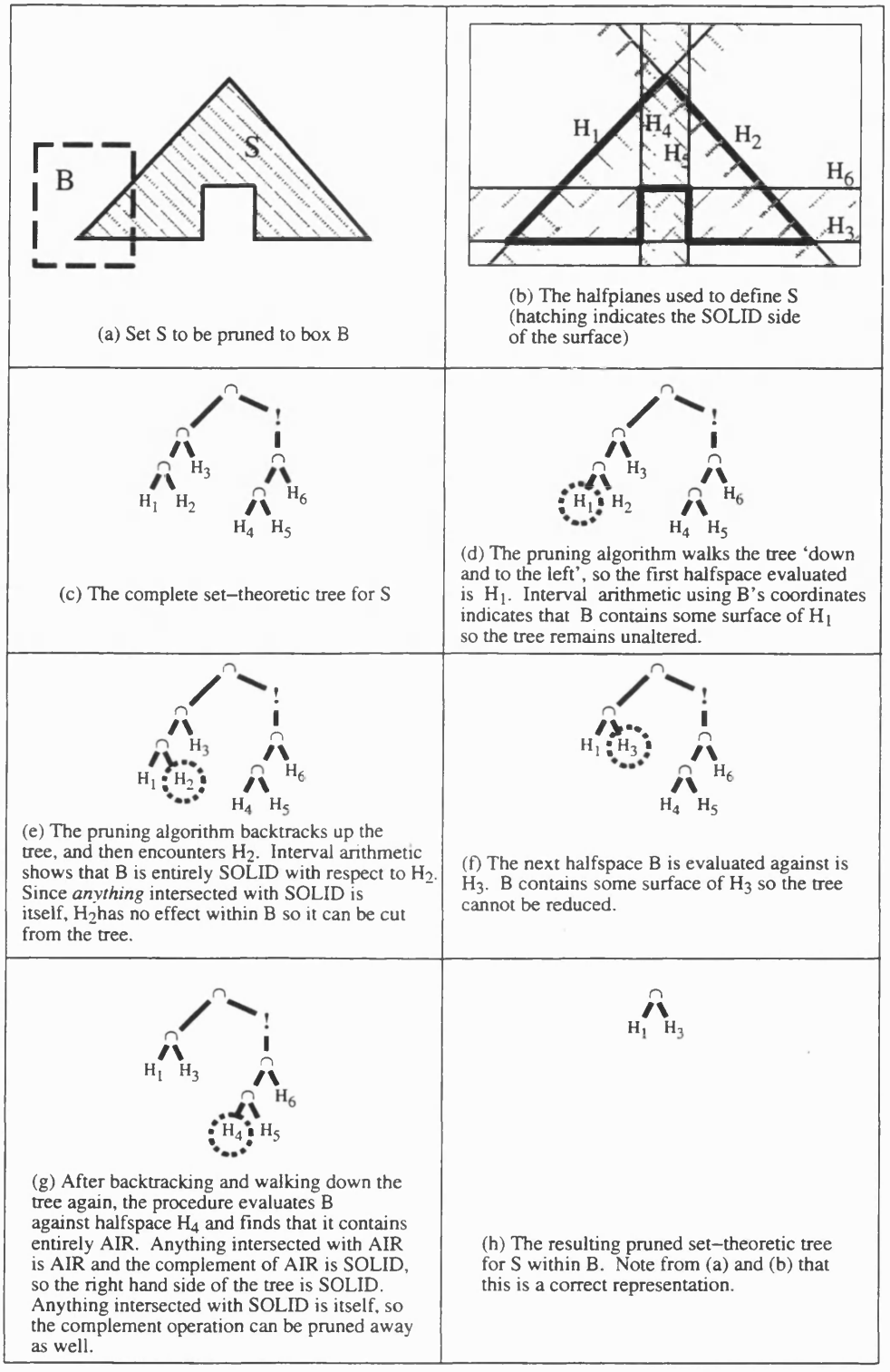


Figure 3.4: **Pruning a set-theoretic tree to a box.**

division (BSD), whereby:

1. The sets in a model are pruned to the model's box.
2. If any of the sets are still considered too complicated and the box is larger than a pre-defined minimum size the box chopped into two along some axially-aligned plane.
3. The model's sets are placed into each of the two boxes created, to make two children models, and the process recurses.

Before such division begins, the user defines what constitutes a set being 'too complicated' and how small a box gets before it is considered to small to chop again. A method for deciding where the axially-aligned chop is positioned must also be defined—a simple decision procedure is to chop the longest dimension in half.

Note that an alternative to binary division is 2^n division, whereby in two-dimensions the model is chopped into four cells and in three-dimensions it is chopped into eight. A tree representing a space divided by the 2^n scheme will typically have less internal nodes than its binary-tree equivalent (since several chops effectively occur simultaneously), but more leaf nodes (since it has less freedom to adapt to the contents of the space). Since svLis models typically have leaves which are large in memory terms compared to the internal nodes, binary division is preferred. For svLis-m, where the leaves are multidimensional models which are more complicated, the advantage of binary division is even greater. The memory requirement of the two schemes in multidimensional cases is discussed further in Wise [103].

3.3 Hypersvlis

3.3.1 Motivation

The omnimodel concept to be described in Chapter 5 originated as the basis of a novel method of feature recognition which was first devised by John Woodwark. To implement this method Parry-Barwick and Bowyer [74] developed hyper-svLis [75, pp168–178], a pseudo-nine-dimensional extension to svLis specifically

designed to represent the three dimensions of ordinary space and six dimensions corresponding to the degrees of freedom of an unconstrained three-dimensional object.

hypersvLis is not truly nine-dimensional since it only stores ordinary three dimensional svLis sets (the multidimensional effect is achieved through special enquiry functions as described below). It did, however, play an important role in the initial research into applications of multidimensional set-theoretic modelling.

3.3.2 Dimensions

Since hypersvLis was developed specifically to represent what I refer to as an omnimodel (Chapter 5), its nine dimensions are hard-coded to represent the three dimensions of ordinary space and the six degrees of freedom of an unconstrained three-dimensional object (three for translation and three for orientation implemented using Euler angles).

3.3.3 Points and boxes

hypersvLis points and boxes explicitly store values in the nine dimensions mentioned above.

3.3.4 Linear halfspaces, other primitives and sets

hypersvLis does not provide classes for halfspaces, other primitives or sets—it uses ordinary svLis objects, attaching a label to each object to indicate if it is static or has degrees of freedom. The multidimensional effect is achieved via the enquiry procedures, such as those which membership test a nine-dimensional point or calculate the potential value range for a nine-dimensional box. If an object is labelled as static, these procedures ignore the values in the dimensions relating to degrees of freedom and test the remaining three dimensional point or box using the ordinary svLis procedures. If, however, a set is labelled as 'moving', the signed distance function for each halfspace is evaluated by a special procedure

which takes into account the rotation and translation transformations which the degrees of freedom perform on the original halfspace[1]. These halfspace procedures occur at the leaves of the primitive and give an answer formulated in terms of all nine dimensions of the point or box. The results are then fed back up the primitive and set trees according to the rules described in Sections 3.2.5 and 3.2.7 respectively, and the effect is to supply enquiries about multidimensional sets in an omnimodel.

3.3.5 Models, interval arithmetic, pruning & recursive division

A hypersvLis model associates a list of sets (each labelled as static or moving) with a nine-dimensional hyperbox. The enquiry functions described in the previous section enable interval arithmetic and pruning to be used in an identical manner to svLis, which in turn enables recursive division (although, as discussed later, the cost of division can increase exponentially with the number of dimensions).

3.4 svLis-m

3.4.1 Motivation

hypersvLis was used to implement not only Woodwark's method of feature recognition, but also novel methods for obtaining images of Minkowski sums and for nesting arbitrary shapes (Parry-Barwick [75]). Further application areas for omnimodel analysis and general multidimensional modelling also became apparent but the hard-coded nature of hypersvLis did not fully exploit the multidimensional property of set-theoretic modelling and limited it to a subset of these applications. Thus, Adrian Bowyer, David Eisenthal and I developed a new multidimensional modeller called svLis-m (short for svLis-multidimensional).

[1]These transformations are detailed in Chapter 5 which describes how they are handled using svLis-m.

svLis-m uses the same modelling approach as svLis (with slight variations, see below) and can be used in conjunction with its three-dimensional relative; svLis models can be used as input and then swept into additional dimensions, and multidimensional svLis models can be sliced down to three or less dimensions and fed to svLis (to obtain graphical output, for example). However, svLis-m can be used as a stand-alone kernel.

3.4.2 Dimensions

Dimensions are represented by a svLis-m class which associates with each one a name, description, and any other information the user wants to attach. Dimensions are created as and when the user requires them, up to a limit which is currently set as 32 (raising this upper limit requires only one class to be changed, but would result in additional computational cost for each operation).

Note that it is often useful to regard svLis-m as an algebra system in which the dimensions are the variables. Indeed, the line between an algebra system and a geometric modeller is a hazy one: however we regard svLis-m as on the latter side due to its exploitation of the linear halfspace basis (p. 66), its explicit representation of spatial locality using boxes and its use of automatic simplification via pruning.

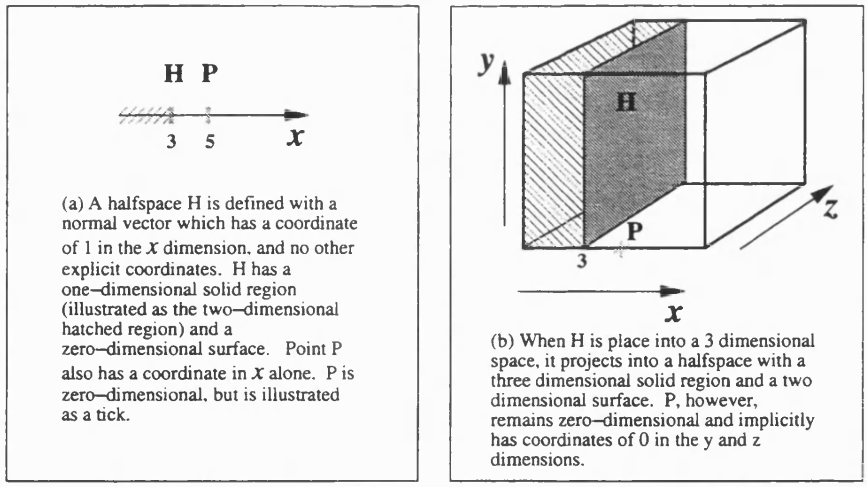
3.4.3 Points and boxes

The svLis-m point/vector class can explicitly store a coordinate value for any number of the existing dimensions (including zero). Since a point is a zero-dimensional entity regardless of the space in which it is embedded, a point embedded in a space which contains dimensions which are not explicitly represented in the point implicitly takes a coordinate of zero in each of those unspecified dimensions. Thus, a point which does not have an explicit coordinate in any dimension is the origin regardless of which space it is placed in. Boxes behave in a similar manner—they can explicitly store an interval in any number of the existing dimensions, and will implicitly have the interval $[0, 0]$ in every dimension that is not explicitly represented.

3.4.4 Linear halfspaces

As with svLis (p. 64), a linear halfspace is represented as a normal vector (which is normalised) and an offset value. Like any point or vector, the normal vector to a linear halfspace may have any number of explicitly specified coordinates so, for example, a linear halfspace might only have an x coordinate (which normalisation will force to have magnitude 1), even if it is going to be embedded in a three dimensional space. This makes sense since unlike points, the dimensionality of a halfspace is not fixed—it depends upon the dimensionality of the space in which it is embedded. For example, if a third dimension was introduced to the space illustrated in Figure 1.3 (a) (p. 23), the disc halfspace would extrude into a cylindrical halfspace whose axis is parallel to the third axis. Likewise, if a linear halfspace which has a normal vector with an explicit coordinate in x alone is embedded in x, y, z space, it will project parallel to the y and z axes to give a perfectly well-defined linear halfspace in three dimensions. In mathematical terms, the addition of a dimension causes a halfspace to project into the Cartesian product of the original halfspace and the extra dimension.

The effect of placing halfspaces and points into a higher dimensional space is illustrated in Figure 3.5.



(a) A halfspace H is defined with a normal vector which has a coordinate of 1 in the x dimension, and no other explicit coordinates. H has a one–dimensional solid region (illustrated as the two–dimensional hatched region) and a zero–dimensional surface. Point P also has a coordinate in x alone. P is zero–dimensional, but is illustrated as a tick.

(b) When H is place into a 3 dimensional space, it projects into a halfspace with a three dimensional solid region and a two dimensional surface. P, however, remains zero–dimensional and implicitly has coordinates of 0 in the y and z dimensions.

Figure 3.5: **The effect of placing a halfspace and a point into a higher dimensional space.**

3.4.5 Other primitives

Unlike svLis, svLis-m does not have a primitive class which is distinct from the set class—instead, svLis-m supports arithmetic and transcendental operators on sets in order to remove svLis' restriction that arithmetic operators cannot occur above boolean operators in the tree. Although it is difficult to imagine how such freedom could be exploited[2] svLis-m was designed with maximum flexibility in mind on the basis that new applications may continue to become apparent with unpredictable requirements.

Note that svLis-m does not supply standard primitives or set equivalents. Although it is possible to define n-dimensional versions of primitive shapes (an n-dimensional sphere is trivial to define, for example) each instance would be dimension specific. As discussed in the previous section, svLis-m halfspaces project into their Cartesian product if they are placed in a higher dimensional space—thus an object originally defined as a sphere would become a cylinder and the special tag attached to it would become misleading.

3.4.6 Sets

As mentioned above, svLis-m sets incorporate the functionality of primitives. Since union and intersection operations introduce discontinuities to the surface of a set, any set which contains such an operator has an undefined gradient. For those sets which contain only arithmetic and transcendental operators (equivalent to a svLis primitive), the gradient consists of a set for each dimension that has a value explicitly set in a normal of a halfspace in the set tree.

Apart from this voluntary departure, the representation of multidimensional sets within svLis-m is identical to svLis—fully exploiting the dimension-independent property of the set-theoretic notation.

[2]Unfortunately at the time of publication images of the results cannot be obtained since svLis-m relies on svLis for graphical output. However, since a ray-tracer has now been developed for use with C-space exploration, images should be available in the near future.

3.4.7 Models, interval arithmetic, pruning & recursive division

As with svLis, A svLis-m model is a list of sets which is associated with a box which defines a region of interest. Interval arithmetic, pruning and recursive division are implemented in an identical manner to that described in Sections 3.2.9 and 3.2.10 and are used extensively to break complicated models into simpler pieces.

3.4.8 'Thin' sets

Like svLis, svLis-m supplies an 'abs' operator which, by forcing the potential field positive everywhere except the surface, makes a set 'thin' (of zero thickness). This operator can be applied to any set.

3.4.9 Slicing svLis-m objects

svLis-m objects can be sliced to any lower dimensionality by fixing each unwanted dimensions at a value, then replacing all references to that dimension by the fixed value. For example, consider a model consisting of the simple six-dimensional set represented in Figure 3.6 in a box with intervals $([\mathbf{x} : -1, 1], [\mathbf{y} : -1, 1], [\mathbf{z} : -1, 1], [\mathbf{x}' : -1, 1], [\mathbf{y}' : -1, 1], [\mathbf{z}' : -1, 1])$.

This model can be sliced to a three-dimensional model by fixing three of the dimensions at specific values—for example Figure 3.7 shows the set if it is sliced at the point $(\mathbf{x}' : 1.8, \mathbf{y}' : 0.3, \mathbf{x}' : -0.4)$.

The sliced model consists of this set inside a box with intervals $([\mathbf{x} : -1, 1], [\mathbf{y} : -1, 1], [\mathbf{z} : -1, 1])$. Note that the sliced box, unlike the sliced set, is independent of the coefficients of the slice-point—only the dimensions of that point have an effect.

In this illustration, the six dimensional set is a translational omnimodel (Chap-

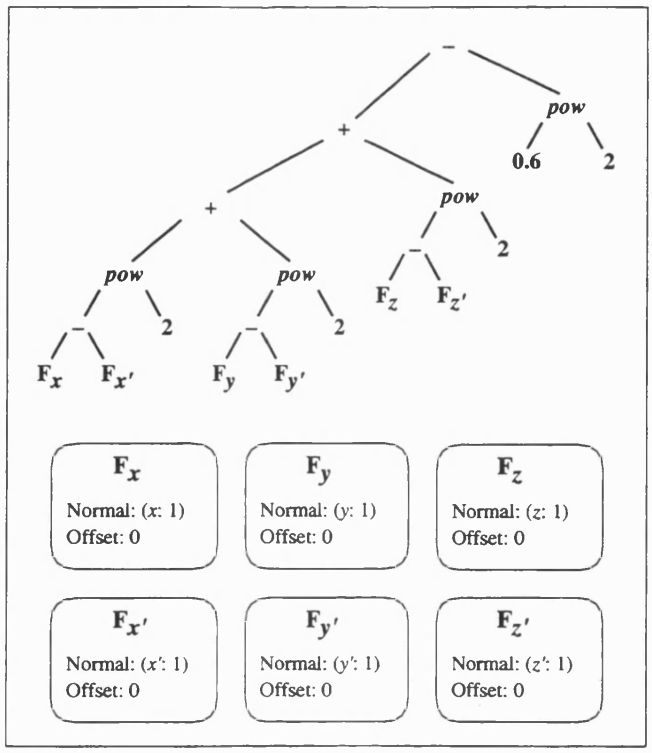


Figure 3.6: **The set-tree of a six-dimensional set**

ter 5) representing a sphere which was initially centred on the origin. The sliced set is a copy of the original, translated by the values of the slice point. Indeed, given the values illustrated, the sliced model will be the empty set since for those translation values the sphere has left the region of 'world' being examined.

Here, slicing had an intuitive and uninspiring effect. However, with a more subtle choice of slice-point it can lead to interesting and useful results as will be seen later—especially in Chapter 8 (C-space mapping for a manipulator arm). It also enables three dimensional models to be obtained which can be fed to svLis such as for graphical display.

3.5 Summary & conclusions

This chapter has outlined the set-theoretic modelling approach adopted by the Geometric Modelling group at the University of Bath. The two key features of this approach, are:

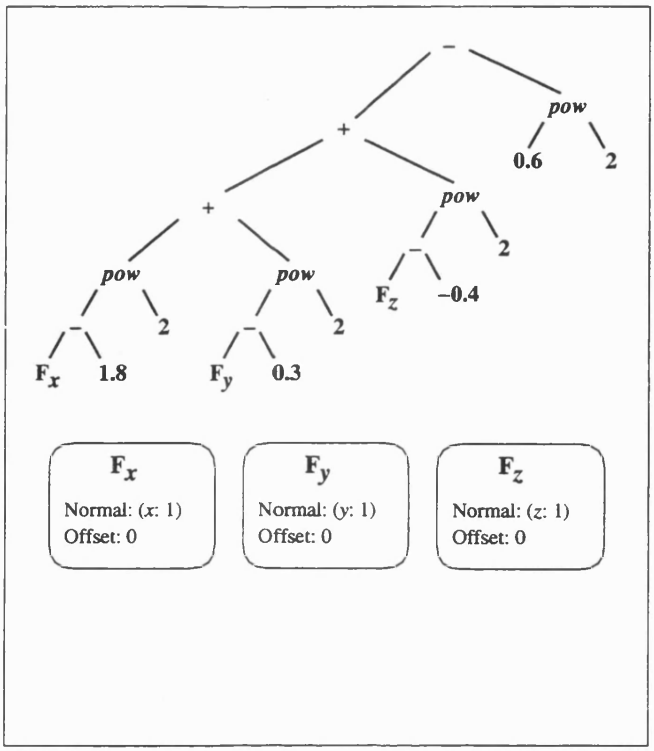


Figure 3.7: **A six-dimensional set sliced to three dimensions**

- Use of the *linear halfspace basis* which represents all primitives as expressions in terms of linear halfspaces—any solid with an implicitly defined surface can be represented in this way. Our modellers support some transcendental operators in addition to arithmetic and boolean operators.
- Extensive use of interval arithmetic, which enables sets to be *pruned* to an axially aligned box. This in turn enables recursive division to systematically break a complicated model into more simple pieces.

Since both of these extend directly to any number of dimensions, we have developed svLis-m—a set-theoretic modelling kernel which can represent objects of near-arbitrary dimensionality. svLis-m has been designed to be a general geometric modelling tool and to be useful as a stand-alone kernel, independent of svLis. However it can be used in conjunction with its three dimensional relative—for example svLis objects can be used as input and svLis' graphics utilities can be used to examine slices of multidimensional sets.

Chapter 4

`Orienteer`—a tool for C-space map validation

In order to enable even multidimensional C-space maps to be validated, I developed a C-space map exploration tool, called `Orienteer`, which acted as a graphical user interface to the svLis-m C-space maps produced by the mapmaking algorithms. Figure 4.1 shows a snapshot.

Features of `Orienteer` include:

- Any system of two- or three-dimensional bodies can be displayed (subject to the maximum number of dimensions allowed by svLis-m, currently 32).
- Each nomad can be controlled by either
  - A 'DOF controller' panel, with which the user can specify either an absolution configuration or a relative step.
  - A six-degree-of-freedom controller (we used a Magellan SpaceMouse [61]) which can be used to control any combination of the nomad's degrees of freedom.
- A 'Membership test' mode in which, before the display is updated to represent a specific configuration, that configuration is tested against the corresponding C-space map. The classification (*safe*, *prohibited* or *contact*) is displayed.

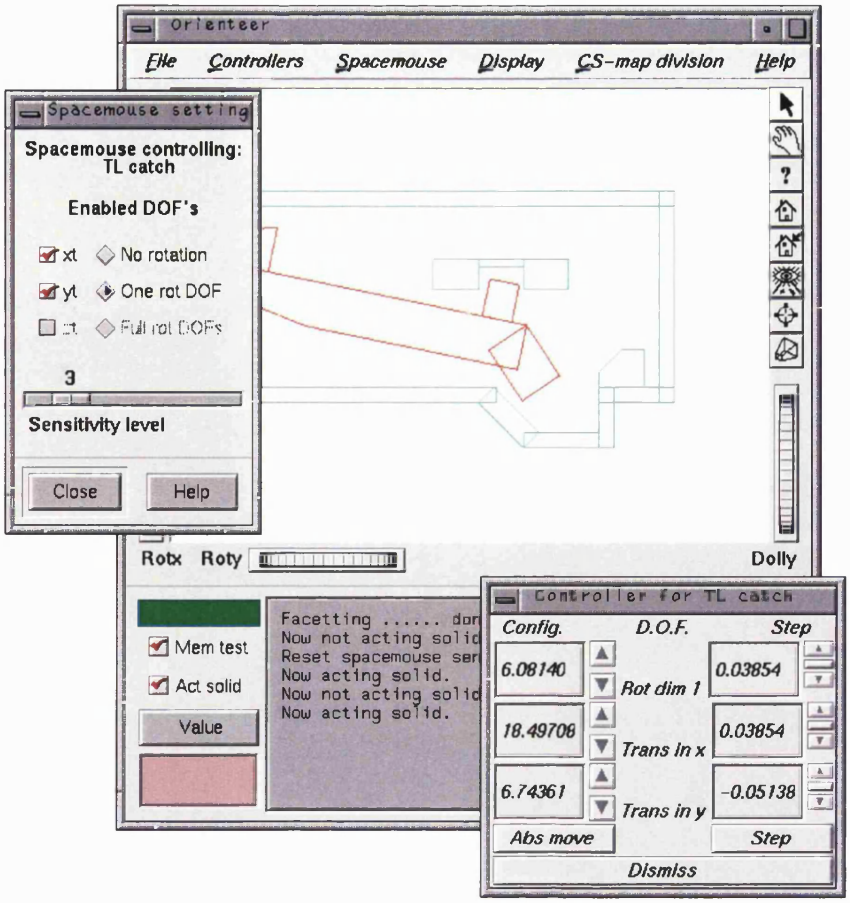


Figure 4.1: **A snapshot of the `Orienteer` C-space validation tool**

- An 'Act solid' mode in which the display will only be updated to a specified configuration if that configuration is classified as *safe* or *contact* in the C-space map.

Thus, a C-space map for a mechanism can be validated by loading the geometry of the mechanism and the C-space map into `Orienteer`, moving the parts into an interesting region of the C-space map (perhaps by temporarily disabling the 'Act solid' mode, in order to move to a disconnected component of free-space) and then observing if the objects 'hit each other' and stop moving in the appropriate configurations.

Chapter 5

Creating an approximate C-space map via projection of an 'omnimodel'

5.1 Introduction

This chapter describes how, for any system of nomads, a static multidimensional model can be constructed which represents every interaction that occurs as the nomads exercise their degrees of freedom. It then discusses five algorithms which orthogonally project such a model (referred to as an *omnimodel*) into the C-space to obtain an approximate global map of the C-space, namely:

- Collision detection at discrete configurations (`collDetPoint`)
- Collision detection for resolution-sized C-space boxes (`collDetBox`)
- Collision detection for binary spatially subdivided C-space boxes (`collDetBSD`)
- Isotropic binary spatial division (`isotropicBSD`)
- 'Learning' collision detection for recursively-subdivided C-space boxes (`L-collDetBSD`

5.2 Degrees of freedom as dimensions

The omnimodel concept originates from a method of feature recognition devised by John Woodwark and implemented by Adrian Bowyer and Stephen Parry-Barwick [74]. The key idea is to represent each mechanical degree of freedom of a system of rigid parts as an additional geometric dimension and to combine this with the geometry of the parts. This is best explained by initially considering one-dimensional models in which solid objects are closed intervals along an axis, $\mathbf{x}$, say.

Consider the nomad and obstacle sets shown in Figure 5.1 (a). When the nomad is placed in its initial configuration in the space of the obstacles, it results in the one-dimensional model shown in Figure 5.1 (b).

The omnimodel concept introduces to this model a second dimension which, instead of being independent to the first, *represents the motion of the nomad along its translational degree freedom.* The resulting two-dimensional model is shown in Figure 5.1 (c). As the value of the $\mathbf{x}$-translation dimension ($\mathbf{x}'$) increases, the nomad slides in the direction of increasing $\mathbf{x}$, sweeping out the diagonal region. In contrast, the position of the obstacle is unaffected by the translation of the nomad, so it projects orthogonally in the new dimension.

The model shown in Figure 5.1 (c) is an *omnimodel* and contains information regarding every interaction that occurs as the nomad exercises it degree of freedom between the values $\mathbf{x}'_{\mathbf{lo}}$ and $\mathbf{x}'_{\mathbf{hi}}$. Omnimodels can be analysed for a number of purposes, including its original application, feature recognition—for details of recent work in this area see Eisenthal [27].

> *The genesis of this thesis was my observation that if the intersection of the swept nomad and the obstacle is projected parallel to the* $\mathbf{x}$ *dimension, the resulting intervals in the* $\mathbf{x}'$ *dimension are the configuration space obstacles that the obstacle causes to the nomad.*
>
> This is illustrated in Figure 5.1 (d).

For the one dimensional example shown, computing the configuration-space obstacles directly from the original models is trivial. However the method of creating

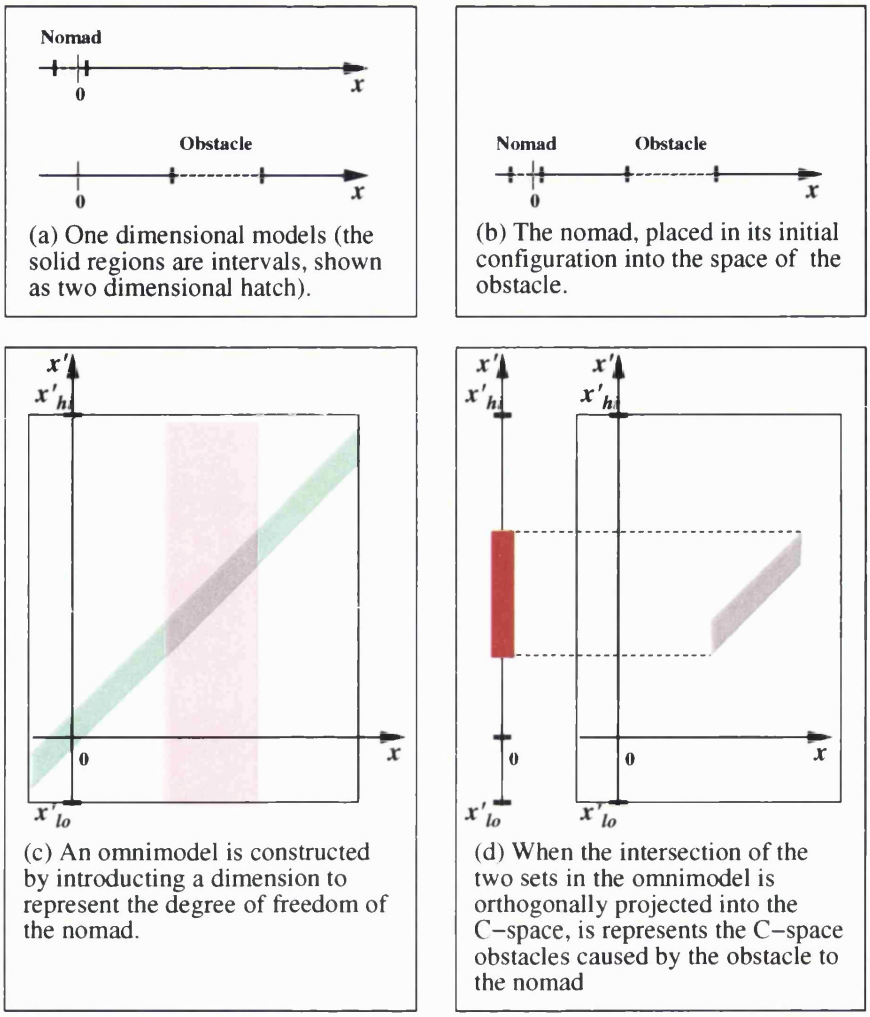


Figure 5.1: **Illustration of an omnimodel for a one-dimensional system.**

an omnimodel and projecting the intersection into the configuration space is inherently general and can be applied to much more difficult problems. Two- and three-dimensional systems of parts can be handled, and an additional dimension can be incorporated to represent each degree of freedom of the system—with rotations handled in a similar way to translations.

Note that for simplicity, this chapter refers to a single nomad throughout—however, the omnimodel method works in principle for systems of any number of nomads of the same dimensionality. The workspace dimensions are shared by all nomads, whilst each additional degree of freedom of a nomad adds an additional dimension—so an omnimodel representing two three-dimensional nomads with full degrees of freedom would be fifteen-dimensional. Where multiple nomads are

involved, the C-space is obtained by projecting all regions of intersection between objects. Thus the set which is examined each time an omnimodel is pruned is the union of the pairwise intersections between objects.

5.3 Omnimodel construction

For a b-dimensional system of rigid parts with a total of c degrees of freedom, the omnimodel has a $(b + c)$-dimensional box. The bounds on the workspace dimensions must be large enough to contain the region in which interactions will occur; the bounds on the C-space dimensions (which define the configuration space) limit the degrees of freedom of the nomad(s).

Into the omnimodel box are placed the obstacle set (which may consist of multiple obstacles unioned together) and a list of nomad sets. The obstacle set is merely a svLis set which has been converted to svLis-m; the halfspaces at the leaves will have normals with explicit values only in the workspace dimensions so, as described in Section 3.4.4, these orthogonally project into $(b+c)$ dimensions when they are placed into the omnimodel box. In contrast, each nomad set must be swept into the additional dimensions such that it represents the nomad exercising its degrees of freedom.

Remember that we represent every set as a tree which has constants and linear halfspaces at the leaves (Sections 3.2.4– 3.2.7). This use of the linear halfspace basis allows us to sweep nomads of arbitrary complexity simply by sweeping each linear halfspace at a leaf—the tree of arithmetic, transcendental and boolean operators which defines the the shape of the object remains constant. Omnimodel construction therefore requires the creation of multidimensional ruled surfaces corresponding to swept linear halfspaces.

5.3.1 Incorporating translational degrees of freedom

In order to introduce a dimension ($\mathbf{x}'$) to represent translation in workspace dimension $\mathbf{x}$, each halfspace leaf of the form $A\mathbf{x} + B\mathbf{y} + C\mathbf{z} + D \leq 0$ is replaced

with a tree with the following form (which corresponds to the rigid transformation matrix for a translation):

$$A(\mathbf{x} - \mathbf{x}') + B\mathbf{y} + C\mathbf{z} + D \leq 0 \qquad (A, B, C, D \textit{ const.})$$

This is a four-dimensional linear halfspace which sweeps diagonally in the new dimension like the two-dimensional nomad in Figure 5.1 (c) (p. 84).

In the same way, additional degrees of freedom can be introduced to represent translation in $\mathbf{y}$ and $\mathbf{z}$. These transformations can be performed in any order, and any combination of the three possible degrees of freedom can be obtained.

5.3.2 Incorporating rotational degrees of freedom

Rotation in the plane

Consider a linear halfspace embedded in xy-space (Figure 5.2 (a)) which is represented by an implicit polynomial of the form:

$$A\mathbf{x} + B\mathbf{y} + D \leq 0 (A, B, D \textit{ const.})$$

If a third dimension (θ) is introduced to represent the rotation of that plane about the origin, the offset of the halfspace from the origin (D) remains constant, whilst the surface normal vector (A_θ, B_θ) at any value of θ can be obtained by applying the matrix for rotation in the plane to the surface original normal vector (A, B). The result is:

$$(A\cos\theta - B\sin\theta)\mathbf{x} + (A\sin\theta + B\cos\theta)\mathbf{y} + D \leq 0$$

Thus, in order to give a two-dimensional nomad set one rotational degree of freedom about the **z**-axis, each leaf halfspace is replaced by a tree of the form illustrated in Figure 5.2, which shows the transformation on a particular planar halfspace (or flat) F. Note that each leaf of the sweep set is still either a constant or a linear halfspace: the constants are pulled out from the original halfspace representation whilst the other leaves ($F_{\mathbf{x}}$, $F_{\mathbf{y}}$ and F_{θ}) are half-spaces with normals parallel to the **x**, **y** and θ axes respectively.

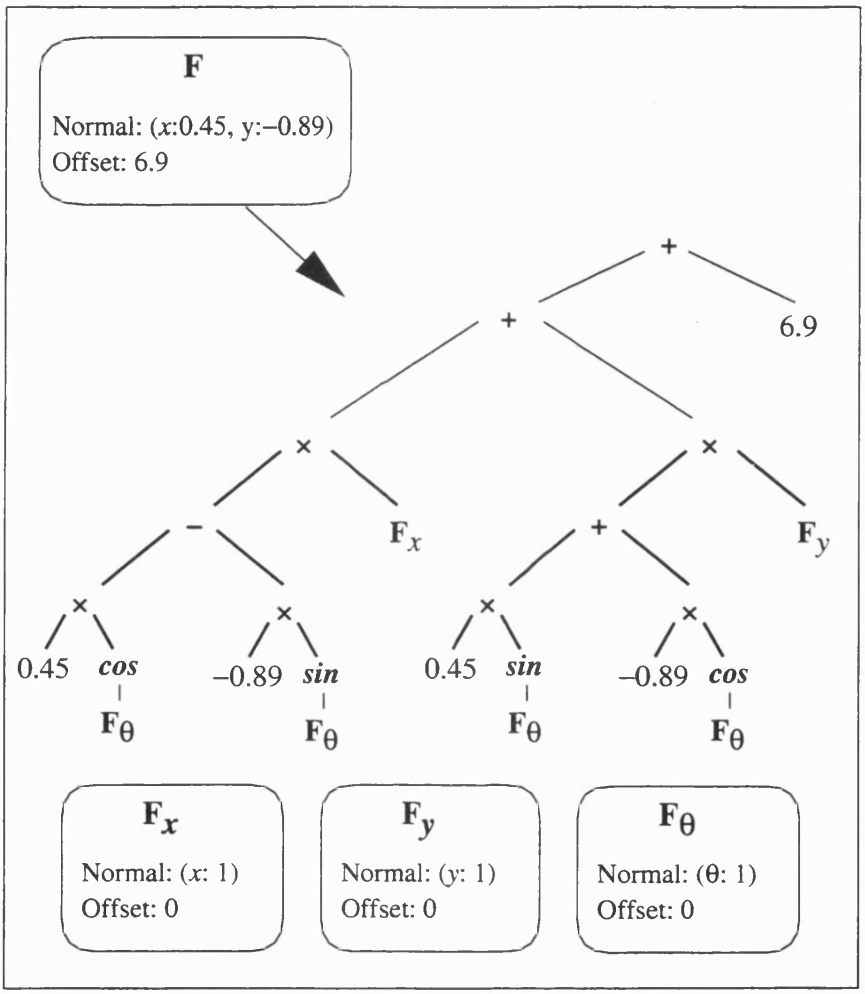


Figure 5.2: **Introducing rotation to a two-dimensional linear halfspace**

A plane swept in this way produces a helicoid, like the one illustrated in Figure 2.8 on page 39.

Parameterising rotation in three-dimensional space

Rotational freedom in three dimensions is typically parameterised using Euler angles, quaternions or rotation vectors. I chose to implement the omnimodel methods (and thus all my algorithms) using Euler angles since those parameters correlate more intuitively to degrees of freedom than the axis and angle parameters of the other schemes. Note however that despite the conceptual difficulties, all of the algorithms described in this thesis could be implemented using quaternions—indeed, such an implementation is discussed as future work in Chapter 9. Of the various flavours of Euler angles I chose to use Roll-Pitch-Roll since a sample of robotics textbooks (Spong and Vidyasagar [94], Paul [77], N-Nagy and Siegler [71], Snyder [92], Selig [89]) suggested that to be the most common. Thus, halfspaces are given full rotational freedom by introducing three dimensions, ϕ (corresponding to rotation about the original **z** axis,), θ (corresponding to rotation about the new **y** axis after the first rotation) and ψ (rotation about the **z** axis after it has been transformed by the first two rotations). This is illustrated in Figure 5.3.

The rotational configuration space (the set of all orientations) is bound by the box $\phi : [0, 2\pi), \theta : [0, \pi), \psi : [0, 2\pi)$. The θ interval has an upper limit of π instead of 2π to reduce redundancy—if the full range was covered, every configuration could be achieved using at least two different combinations of parameters. Note that despite this precaution, the scheme still includes some redundancy in that if θ is zero, ϕ and ψ correspond to rotations about the same axis, so any combination of ϕ and ψ values with the same sum result in an identical orientation. In implementations of Euler angles this problem is handled by forcing ψ to zero whenever θ is zero such that all rotation is parameterised by ϕ.

It should be noted then, that this implementation of rotation only allows two types of rotational freedom to be represented efficiently—either rotation in the **xy** plane, or a full three degrees of freedom. Not only is rotation about an arbitrary axis impossible to represent, rotation about the x-axis cannot be represented by a single dimension since it can only be achieved by a combination of rotation about the z and y axes; and rotation about the y axis cannot be represented by the θ parameter alone, since that parameter is restricted to $[0, \pi)$.

This restriction is one reason why alternative parameterisations are discussed in

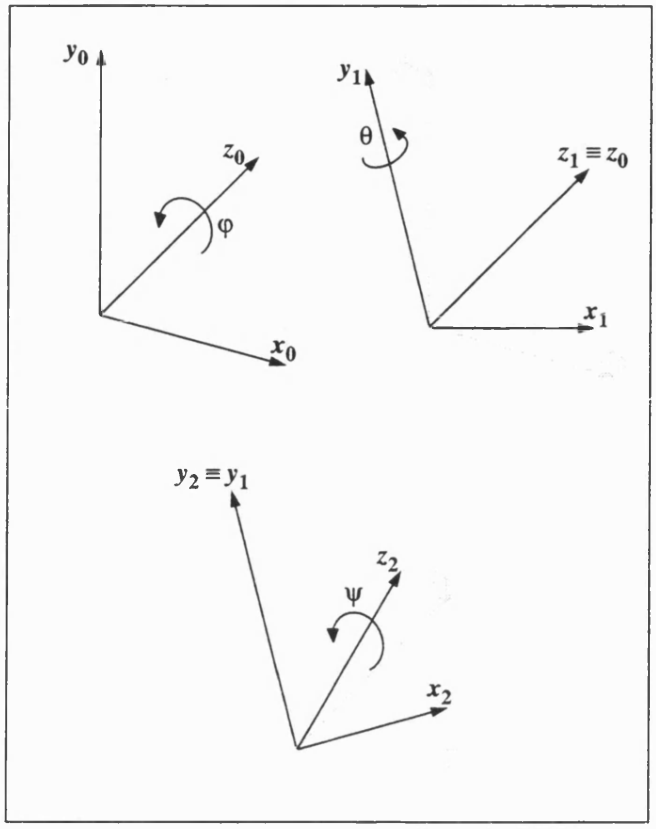


Figure 5.3: **Roll-Pitch-Roll Euler Angles**

Future Work (Chapter 9).

Introducing three-dimensional rotation to an omnimodel

When a plane is swept into a six dimensional space, where the extra dimensions correspond to these Euler angles, it sweeps out a five-dimensional ruled surface, which might be considered a 'hyper-helicoid'. Although this surface is difficult to visualise, its equation is obtained in the way described above for rotation in the plane: the offset remains constant whilst the surface normal is obtained by multiplying the original normal vector by the rotation matrix formulated in terms of Euler angles:

$$\begin{pmatrix} \cos(\phi)\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\psi) & -\sin(\phi)\cos(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\theta)\cos(\psi) \\ \cos(\phi)\cos(\theta)\sin(\psi) + \sin(\phi)\cos(\psi) & -\sin(\phi)\cos(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\theta)\sin(\psi) \\ -\cos(\phi)\sin(\theta) & \sin(\phi)\sin(\theta) & \cos(\theta) \end{pmatrix}$$

The svLis-m implementation of this sweep is worth mentioning for two reasons. Firstly, despite the length of the expanded expression, the code performing the transformation (which is included as Appendix A) is relatively compact. Secondly, the use of reference counting in svLis-m (Section 3.2.5) means that a significant proportion of the tree corresponding to the full expression is not required—repeats of sub-trees are each replaced by a pointer to the original. The latter implementation detail reduces the memory requirement and, when combined with lazy evaluation, significantly reduces the evaluation times.

Note that:

- The rotations about each axis must be performed in a fixed order.
- The rotation transformation must be performed before any translation degrees of freedom are introduced since the transformation is based on the nomad sitting at the origin.

5.4 The effect of division

As described in Section 3.2.10, our approach to set-theoretic modelling makes extensive use of recursive division and pruning. To understand how this is used to analyse the omnimodel it is helpful to consider the effect of chopping the omnimodel in half by splitting one of the sides. If the side split is a workspace dimension (one of $\mathbf{x}, \mathbf{y}, \mathbf{z}$), each child model focuses on interactions which occur in one half of the workspace while the nomad moves with the same amount of freedom. Conversely, if a configuration space side is chopped in half, each sub model contains information about what happens in the same region of the workspace, but when the nomad is restricted to move half as much in the chosen degree of freedom.

5.5 Overview of projection into the C-space

To recap, we have established the following:

1. Using set-theoretic modelling, a multidimensional omnimodel can be constructed which represents every interaction that occurs as a nomad exercises its degrees of freedom. For a one-dimensional workspace, where objects are intervals and the nomad has one degree of translational freedom, the omnimodel can be illustrated as shown in Figure 5.4.

2. If all the parts of the omnimodel where objects overlap are projected orthogonally into the C-space dimensions, the result corresponds to the C-space map for the system being analysed.

3. Using orthogonal division and pruning, an omnimodel can be broken into sub-omnimodels which are typically simpler. Thus, divide-and-conquer methodologies can be employed which recursively divide until sub-omnimodels are reached which are either sufficiently simple, or smaller than a pre-defined resolution.

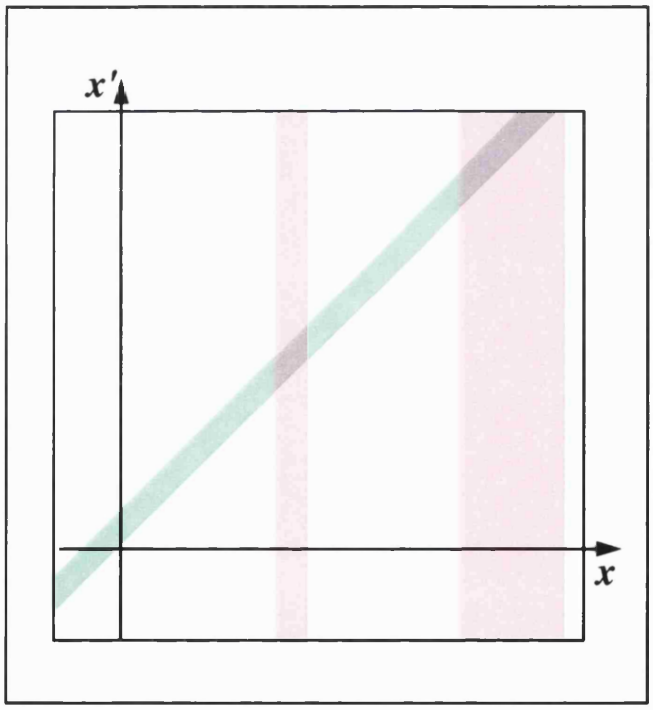


Figure 5.4: **The two-dimensional omnimodel used to illustrate omnimodel projection**

The remainder of this Chapter focuses on five ways recursive division of an omnimodel can be used to obtain an approximate C-space map within which cells are classified as *safe*, *prohibited* or *contact*. In each case, the terminating condition for the divide-and-conquer (that is, the definition of 'sufficiently simple')

is an omnimodel which is either completely *solid* or completely *air* with respect to overlap between objects. Note that Chapter 9, Future Work, discusses the possibility of less dividing and more conquering by combining the algorithms in this chapter with the precise analytical solutions described later (Chapter 6).

5.5.1 Resolution boxes

The C-space maps produced by the algorithms described in this chapter are determined by two resolution boxes:

The C-space resolution box

This specifies the terminating condition for division in the C-space dimensions—i. e. the stage at which division ceases and C-space regions which have not been classified as *safe* or *prohibited* are classified as *contact.* The size and shape of this resolution box will be evident in the final C-space map produced.

For cases involving only translational degrees of freedom, the resolution C-space box will typically have the same length in each dimension; in contrast, for cases involving manipulators with rotational joints, each resolution-sized C-space box would typically have longer sides in dimensions which correspond to higher joints—since reorientation of those joints has less effect on the manipulator position. For cases involving a mixture of translational and rotational freedoms, setting the relative sidelengths is a non-trivial matter discussed in the Test Results section of this chapter and also in Future Work.

The workspace resolution box

The second resolution box specifies the terminating condition for division in the workspace dimensions. For translation-only cases, an obvious default is for this box to have intervals of the same length as the C-space resolution box—but the use of other values is discussed in the Test Results section. For cases involving rotational degrees of freedom, setting an appropriate workspace resolution is a

non-trivial matter and is discussed in Future Work.

5.5.2 Cell merging

Note that three of the five algorithms described below store the C-space map as a binary tree with leaves classified as *safe*, *prohibited* or *contact*. Owing to the nature of the algorithms, each of them produce C-space maps which can benefit from *cell merging*, whereby sibling cells which have the same classification are merged into a single cell. This can take place either during the construction of the map or as a post-processing step. Unless stated otherwise, figures show maps produced with cell merging *disabled* so that more information regarding the construction of the map is evident.

5.6 `collDetPoint`: Collision detection at discrete configurations

5.6.1 Method

Perhaps the most intuitive approach to projecting the omnimodel shown in Figure 5.4 is to imagine a set of rays with their bases in the C-space, spaced apart by the resolution distance, fired across the workspace dimensions. Each ray that hits an overlap between objects results in a resolution-sized *prohibited* box in the C-space, centred upon that ray's base. Conversely, rays which miss the intersection completely result in a *safe* region. *Contact* rays are also possible, since division in the workspace dimension may not be able to separate the nomad and obstacle sets.

When set-theoretic geometry is used, ray-tracing is can be implemented by recursively dividing the ray to establish if and where the ray hits an object. In the case of Figure 5.5 the ray is parallel to the workspace dimension, so the ray can be classified by:

1. Slicing the omnimodel to the configuration point (see Section 3.4.9) to leave a one-dimensional model.

2. Recursively dividing the resulting model in the remaining (workspace) dimension, pruning the intersection set to the box at each step.

3. If a *solid* region is found, the ray must pass through the intersection, so the configuration tested is *prohibited.*

4. If *all* the boxes along the ray are pruned to *nothing*, the ray misses the intersection so the configuration is *safe.*

5. If no *solid* regions are found, but some resolution-sized regions are reached which still have some contents, the configuration is classified as *contact.*

Figure 5.5 illustrates this method of projection, showing the rays which are fired across the omnimodel, and the divisions along those rays which result from binary division in the workspace dimension. In Figure 5.5 (a), the resolutions in both the workspace and the C-space lead to four cells along each side, whilst in Figure 5.5 (a), the workspace resolution is set to a finer value. In both cases, the 1-D C-space map which results is shown on the left.

Note that the change of workspace resolution has a very significant effect—close examination of ray 1 reveals a *solid* piece of intersection, indicating that the ray hits the intersection, so the C-space cell associated with it is classified as *prohibited.* In contrast, closer ray 3 is found to miss the intersection, so that C-space cell is classified as *safe.* This 'more accurate' classification means that a C-space obstacle falls completely between the rays there is nothing in the map to indicate its existence.

When this approach is applied to a case with a two-dimensional workspace, the 'ray' fired at each discrete configuration becomes a two-dimensional plane (parallel to the workspace) which is classified in precisely the same way—slicing followed by recursive division. The same applies for a three-dimensional workspace, where the 'ray' is a three-dimensional cuboid, 'parallel' (or the multidimensional equivalent) to the original workspace, but offset in the C-space dimensions. This is the approach that was used by Parry-Barwick [75] to obtain images of Minkowski sums via an omnimodel.

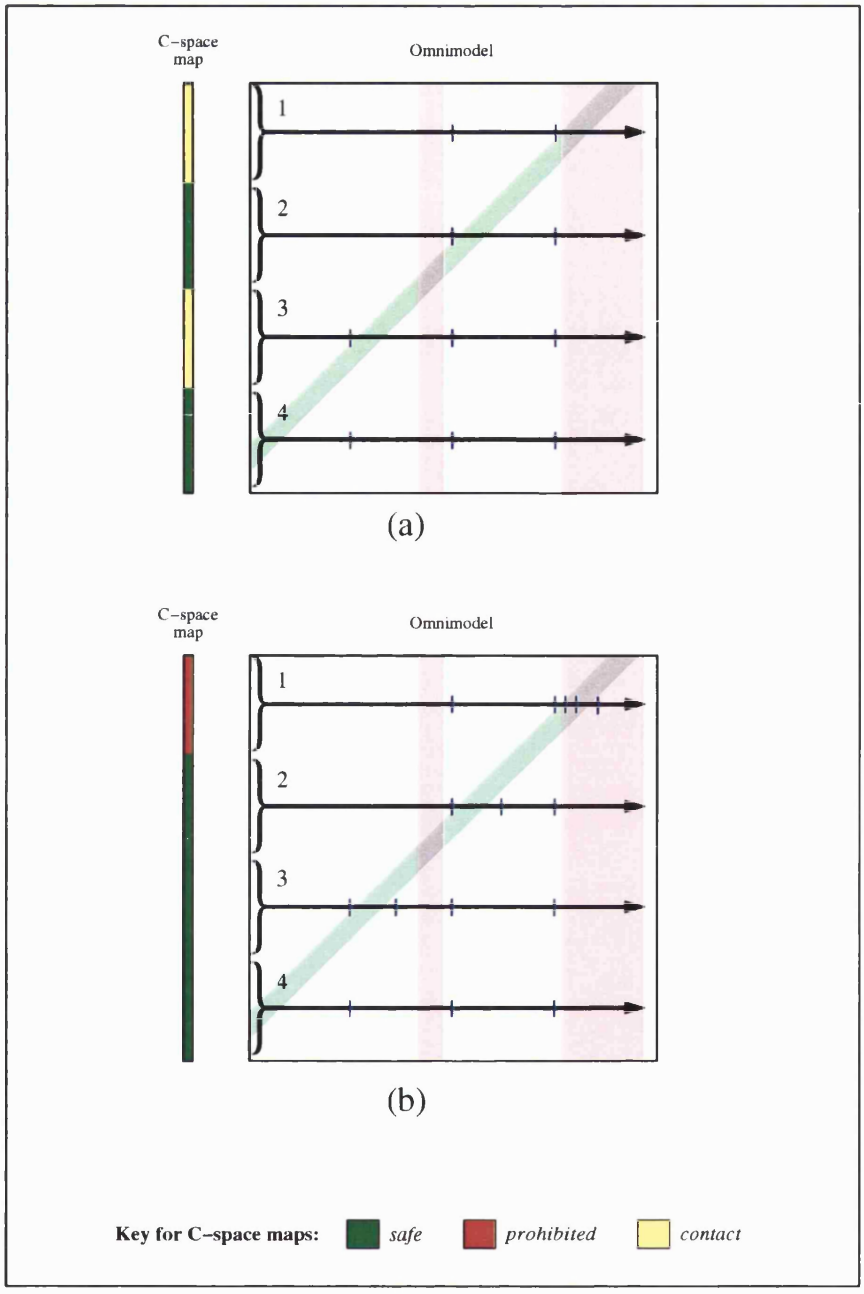


Figure 5.5: **Projecting into a one-dimensional C-space using ray-tracing**

In all cases, the process of classification is exactly equivalent to *collision detection*—the nomad is fixed in a specific configuration and the workspace is recursively divided to establish if any overlap occurs with the obstacles.

5.6.2 Strengths and weaknesses

Though naive, `collDetPoint` does have some strengths:

- Since the boxes being pruned to only have thickness in the workspace dimensions, fewer intervals are involved in the arithmetic. Thus the contact region (that is the layer of contact boxes between the *safe* and *prohibited* regions) is shallower than for the other projection methods. As with all methods, coarsening the workspace resolution makes the proportion of *contact* tend towards 100%. However, for typical parameters the contact region will tend to be just one box deep.
- Since each cell in the resulting C-space map is a uniform size and shape with one of three states, each cell can be represented by one byte or less; if *contact* cells are treated as either *safe* or *prohibited*, only one bit is needed, and memory requirement can be reduced further by employing compression algorithms. This makes a regular grid a relatively compact option for C-space maps which are two-dimensional and or of a low resolution.
- Membership testing is simply a lookup operation in an n-dimensional array so it is extraordinarily fast.

Conversely, collision detection at discrete points suffers from two primary weaknesses:

- Since only discrete points are tested, the result is not conservative. As illustrated above, cells can mistakenly be classified as *safe* when they contain some *prohibited* configurations, and ultimately C-space obstacles can be overlooked altogether.
- An n-dimensional regular grid with a resolution of d cells in each dimension has d^n cells. Although storage requirement might be reduced by compression techniques, d^n collision detection operations must be performed.

5.7 `collDetBox`: Collision detection for resolution-sized C-space boxes

5.7.1 Method

The problem of obstacles falling between rays can be overcome by using additional interval arithmetic to check for collision for a range of configurations instead of discrete configurations. The C-space is divided into a regular grid of resolution-sized boxes, each box is then projected across the workspace dimensions and classified as *safe*, *prohibited* or *contact* by recursively dividing across the workspace dimensions, exactly as before:

1. If an omnimodel is found in which the intersection of obstacle and nomad is *solid*, then every configuration in the C-space box being considered must cause an overlap, so so the whole C-space box can immediately be classified as *prohibited*, regardless of what other regions of the workspace contain.
2. If *all* the omnimodels created by dividing in the workspace are pruned to *nothing*, no overlap can occur, so the C-space box can be classified as *safe*.
3. If no *solid* regions are found, but some resolution-sized regions are reached which still have some contents, the C-space box is classified as *contact*.

Figure 5.6 compares the results from `collDetPoint` and `collDetBox` on a simple case where the obstacle and nomad are identical discs and shows that *contact* region is indeed thicker for `collDetBox` as discussed.

5.7.2 Strengths and weaknesses

The main strengths of `collDetBox` are:

1. Testing C-space boxes instead of discrete configurations achieves the important characteristic of conservatism. In particular this means that even infinitesimal obstacles can be detected since they result in a *contact* region.

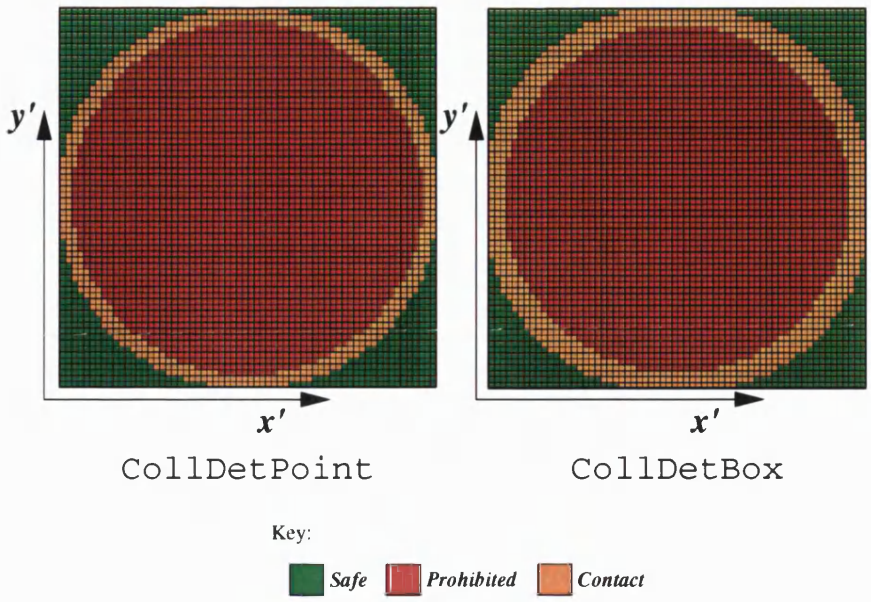


Figure 5.6: **Comparison of results from `collDetPoint` and `collDetBox`**

2. The advantages of fast membership testing, and compact memory requirements for two-dimensional C-spaces, are maintained.

The primary weaknesses of this scheme are:

1. The additional interval arithmetic involved in this method means that, for the same resolution boxes, the proportion of the map classified as *contact* will be greater than for `collDetPoint`. This is also true of the `collDetBSD`, `isotropicBSD` and `L-collDetBSD`, since they produce equivalent maps to `collDetBox`.

2. An n-dimensional C-space at a resolution of d cells in each dimension still requires d^n cells to be stored and, more significantly, d^n cell classifications to be performed.

5.8 collDetBSD: Collision detection for recursively sub-divided C-space boxes

5.8.1 Method

As pointed out above, a regular grid division of an n-dimensional C-space map which is divided to a resolution of d cells along each side results in d^n cells being classified and stored, *regardless of the C-space contents.* However, in all but the most pathological of cases, the configuration space will contain large contiguous *safe* and *prohibited* regions so an adaptive division scheme such as the binary tree or 2^n-tree is often able to store a large proportion of the C-space as a smaller number of larger cells. Indeed, as the division becomes finer, the number of cells in such adaptive division schemes becomes proportional to the size of the $(n-1)$-dimensional surface of the object (Samet [86][p. 10–11]). Thus, employing a binary spatial division scheme instead of a regular grid would enable the number of cell classifications to be reduced from d^n to $O(d^{(n-1)})$.

A recursively subdivided C-space map can be obtained from the following simple algorithm:

Starting with the whole C-space box,

1. Classify the C-space box using the same method as the previous algorithm—recursive subdivision in the workspace dimensions.

   If the workspace resolution is reached and the C-space region cannot be classified as *prohibited* or *safe*, check if the C-space box is smaller than the C-space resolution. If it is smaller, classify the C-space box as *contact*, as before. However, if the current C-space box is larger than the C-space resolution, recurse as follows:

   (a) Divide the C-space region being studied into two—for example, by chopping the omnimodel in half along the longest C-space dimension.

   (b) For each of the two submodels created, recurse by re-entering the algorithm at Step 1, the classification step.

(c) The result for the current C-space box is a binary tree where each child node is the result from mapping a child omnimodel.

The C-space map produced from this algorithm is a binary spatially divided tree, as illustrated in Figure 5.7 (which repeats the equivalent map produced by `collDetBox` for comparison).

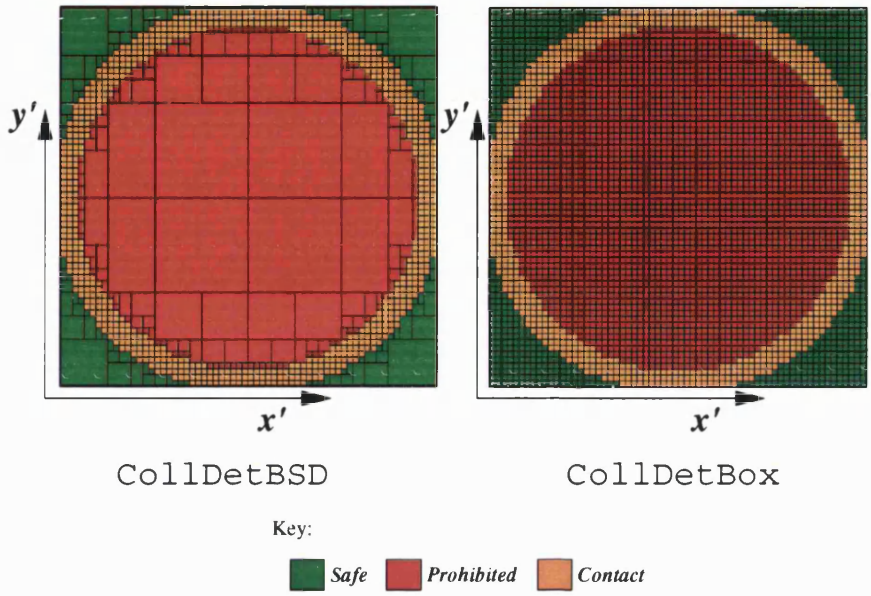


Figure 5.7: **Comparison of results from `collDetBSD` and `collDetBox`**

5.8.2 Strengths and weaknesses

As mentioned above, the advantage of the adaptive division algorithm over the regular grid version is that the number of cells being classified is typically reduced from d^n to $O(d^{(n-1)})$. Since an exponential term is reduced, the expected saving in classification and storage rapidly becomes more important as the dimensionality of the C-space (n) increases. The advantage of adaptive division also increases as the dimensionality of the workspace increases, since classifying each C-space region becomes more expensive, making the time saving due to less cell classifications more significant.

However, a fundamental weakness of the above algorithm is that for each non-leaf node of the result tree, effort is expended dividing the workspace dimensions to classify the C-space box, only to find a classification cannot be made.

5.9 `isotropicBSD`: Isotropic binary spatial division of the omnimodel

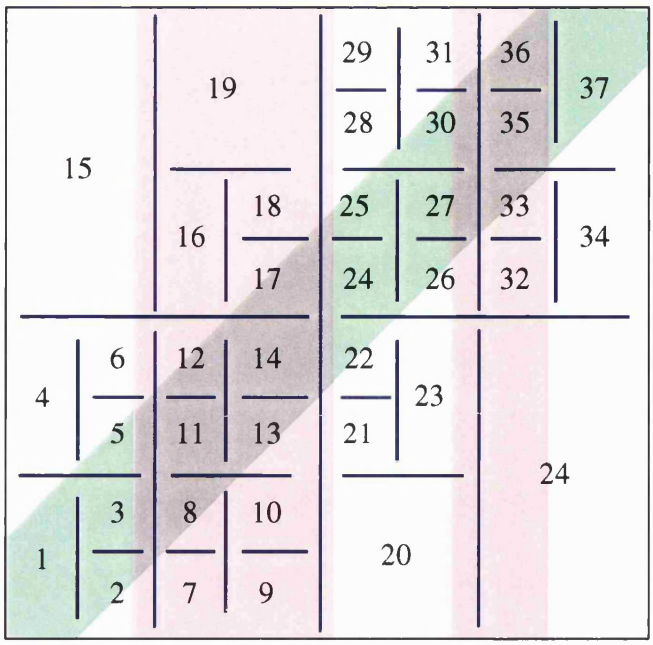
5.9.1 Method

The virtues of adaptive division suggest that instead of alternating between a single division in a C-space dimension and a complete division in the workspace dimensions only, a good projection algorithm could be obtained via a 'pure' binary spatial division of the omnimodel which treated all dimensions equally. In order to do this, the fourth projection algorithm, `isotropicBSD`, works in a fundamentally different manner from the others.

In the other algorithms, each time a C-space classification is made, it is based on what is going on in the whole workspace. Consequently, any point or box in the C-space only needs to be successfully classified exactly once—unsuccessful attempts to classify a parent (larger) box may be made, but as soon as a box has been successfully classified as *safe*, *prohibited* or *contact*, that box is no longer examined. All the boxes which have been successfully classified form the leaves of the divided C-space map.

In contrast, when the omnimodel is divided isotropically, each C-space box which is classified may be classified multiple times, each focusing on a specific region of workspace. Every time the omnimodel is split in a workspace dimension, the algorithm builds two complete maps for the same C-space box and then *superimposes* these maps. The rules for the superimposition are that *prohibited* takes precedence over *contact*, which in turn takes precedence over *safe*.

This is illustrated in Figure 5.8. Examination of cell 6 classifies the C-space region associated with it as *safe*, however when cell 12 (which is concerned with the same region of C-space) is examined, a classification of *contact* is made because a resolution box is reached which is still complicated. The *contact* classification overrides the *safe* classification—however, when cell 14 is examined, that *contact* classification is overridden with *prohibited*. When cell 22 is examined, the same piece of C-space is found to be *safe* again, but that classification does not affect the C-space map.



Cells are numbered in the order they are reached in the bottom–left–first algorithm

Figure 5.8: **An illustration of isotropic binary spatial division**

A naive version of `isotropicBSD` would proceed as follows:

1. Examine the contents of the omnimodel with respect to intersection between objects.

2. If the omnimodel is *solid*, then return a completely *prohibited* C-space box as the result.

3. If the omnimodel is *empty*, then return a completely *safe* C-space box as the result.

4. If the omnimodel contains *surface* of the intersection, check if both the C-space box and the workspace box are smaller than their respective resolutions. If they are smaller, return a *contact* box as the result. If the omnimodel is bigger than resolution size, recurse as follows:

   (a) Divide the omnimodel into two—for example, by chopping the longest side in half.

   (b) Classify each submodel by recursing to Step 1.

(c) If the chopped dimension is a C-space dimension, then, as with previous algorithms, the results from the two sub-omnimodels are independent maps of different C-space regions. The result for the current C-space box is a binary tree, with each child node being a map obtained from a child omnimodel.

(d) If the chopped dimension is a workspace dimension, the C-space maps from the two sub-omnimodels are maps of the same C-space region. The result for the current C-space box is the superimposition of these two maps, obtained by calling a tree-merging algorithm which obeys the superimposition rule described above.

In practise, such an algorithm would be exceedingly inefficient. In particular,

1. Every time the multidimensional omnimodel is chopped in a workspace dimension, two maps of the complete C-space must be merged
2. The superimposition rule (p. 101) means that once a specific C-space box has been classified as *prohibited* by examining one part of the workspace, that C-space region need not be examined again for any other part of the workspace. This is not exploited in the above algorithm.

Both of these inefficiencies are avoided by introducing the notion of a *known map*. Each time the algorithm successfully classifies a C-space box as *safe*, *prohibited* or *contact*, that information is added to the *known map*, which is a record of what has been learned so far. Each time the algorithm divides in a C-space dimension, analysis of each child is based not only on what the omnimodel contains, but also what has been learned about that region of C-space already. Crucially, the algorithm makes sure that at any time, the part of the *known map* corresponding to the current C-space box is available. This approach means that:

1. The *known map* corresponding to the current C-space is instantly checked. If analysis of another part of the workspace has already classified it as *prohibited*, the current omnimodel does not need to be analysed.
2. The algorithm gradually makes local changes to the *known map*, such that the result returned from each omnimodel analysis is a combination of new

results and what was known before. The algorithm no longer merges complete C-space maps.

More specifically, the improved algorithm for `isotropicBSD` is as follows:

1. Initialise the *known map* as a *safe* region which covers the total C-space box, then send it in (along with the complete omnimodel) to the following recursive algorithm:

   (a) Check the *known map* sent in (which always corresponds to the C-space box of the omnimodel sent in) to see if it is all *safe*, all *prohibited*, all *contact* or a divided tree.

   (b) If the *known map* is all *prohibited*, no further analysis of the current omnimodel is required—return the *known map* as the result. Otherwise, continue.

   (c) Examine the contents of the omnimodel with respect to overlap between objects. If the omnimodel is *solid*, then return a completely *prohibited* C-space box as the result.

   If the omnimodel is *empty* of overlap, then no new information has been gained. Return the *known map* (which may be a divided tree) as the result.

   Otherwise, continue.

   (d) If the omnimodel contains *surface* of the intersection, check if both the C-space box and the workspace box are smaller than their respective resolutions. If they are smaller, return a *contact* box as the result. *Note:* the superimposition rule (p. 101) is enforced since if the *known map* had been *prohibited* in this region, the *contact* classification would not have been made (see Step 1b) whilst if the *known map* had been *safe*, it will now be replaced with *contact.*

   (e) If the omnimodel covers a region which has not yet been classified as *prohibited*, and contains surface of the intersection, and is big enough to divide, recurse as follows:

      i. Divide the omnimodel into two by chopping the longest side in half.

ii. If the chopped dimension is a workspace dimension, classify each child omnimodel by recursively entering the algorithm at Step 1a. Both omnimodels will be used to classify the same region of C-space.

For the first child, the *known map* is the same as was sent in for the current omnimodel, so pass that in. However, *analysis of the first child will return a new C-space map, which is a more educated map of the current C-space box.* This map becomes *known map* sent in when the algorithm is re-entered at Step 1a to analyse of the second child omnimodel.

Note that the C-space map obtained from the second child omnimodel will have already taken the *known map* into account, so it will encapsulate everything learned so far.

iii. If the chopped dimension is a C-space dimension, each omnimodel can be analysed independently. However, the *known map* will be different for each—each will get the appropriate half of the current *known map.*

As with the other algorithms, the C-space map for the current omnimodel is a binary tree—each branch is the result from a child omnimodel.

Although it takes a completely different route, this algorithm produces C-space maps which are identical to those produced by collision detection on recursively subdivided C-space boxes, once cell merging (Section 5.5.2) is enabled. This is illustrated in Figure 5.9 which compares output from the two algorithms. Note that maps produced using isotropic division always have cell merging enabled since, in contrast to collision detection on recursively subdivided C-space boxes, this significantly improves the performance. This is because if the *known map* for the current omnimodel is stored as a large *prohibited* box, there is a clear indication that the omnimodel need not be examined; if that *prohibited* region was stored as a divided tree, then either the current omnimodel would be treated as needing examination, or the whole tree would have to be examined to check if all the leaves are *prohibited.*

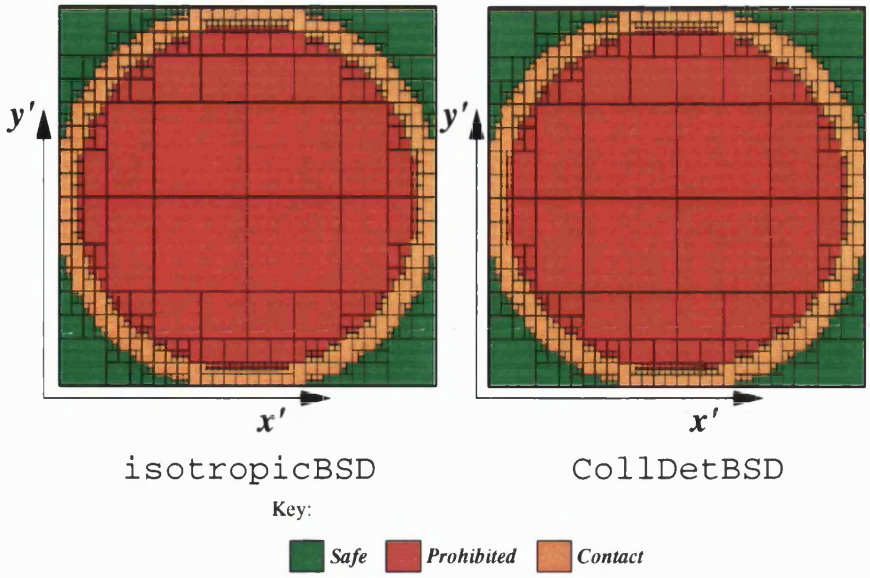


Figure 5.9: **Comparison of C-space maps produced by `isotropicBSD` and `collDetBSD`—note they are identical despite taking completely different routes**

5.9.2 Strengths and weaknesses

In order to compare `isotropicBSD` with `collDetBSD`, consider the problem of projecting the two exaggerated omnimodels shown in Figure 5.10—(a) contains a lot of features which are small compared to the omnimodel box, whilst (b) contains a large contiguous region of overlap.

Figure 5.11 illustrates the division which takes place when `collDetBSD` is used to project Figure 5.10 (a) into the $\mathbf{x}'$ dimension, to coarse resolutions in the workspace and C-space. The fine details of the omnimodel means that every attempt to classify a C-space box by dividing the workspace fails.

Since the number of internal nodes in a binary tree is always one less than the number of leaves, the total number of attempts to classify a C-space box is $(2d^b - 1)$, where d is the resolution along each side, and b is the dimensionality of the C-space. Similarly, the number of nodes in the workspace division tree is $(2d^a - 1)$, where a is the dimensionality of the workspace. Thus, the total number of omnimodels examined is $(2d^a - 1)(2d^b - 1)$.

As Figure 5.12 shows, `isotropicBSD` also fails to make classify any regions of

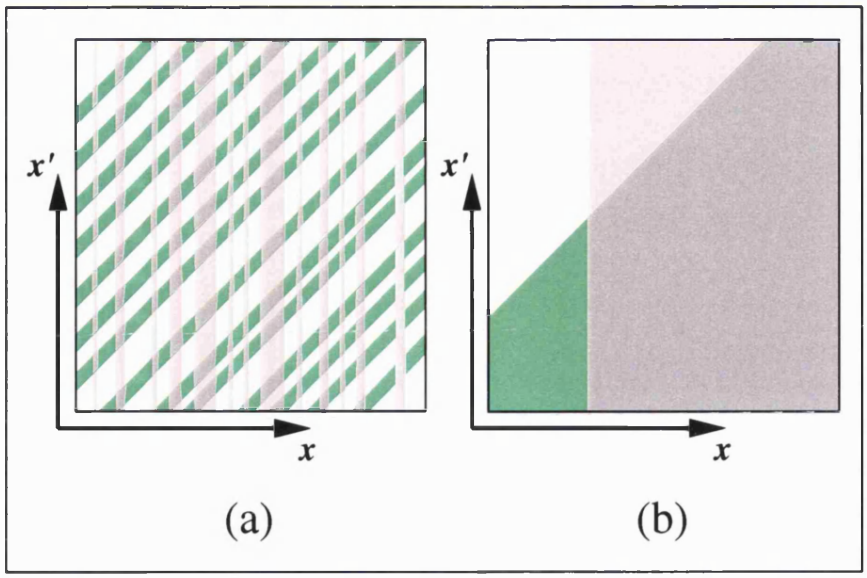


Figure 5.10: **Exaggerated omnimodels which illustrate the strengths and weaknesses of `isotropicBSD`**

the C-space as *safe* or *prohibited*. However, pure division only results in one tree division (dividing in all dimensions), so the total number of omnimodels examined is $2d^{(a+b)} - 1$.

Thus, the ratio of the total number of omnimodels examined in the worst case is

$$\frac{\textit{worst case } \texttt{isotropicBSD}}{\textit{worst case } \texttt{collDetBSD}} = \frac{2d^{(a+b)} - 1}{(2d^a - 1)(2d^b - 1)}$$

For practical values of a, b and d the constant terms are negligible giving:

$$\begin{aligned}\frac{\textit{worst case } \texttt{isotropicBSD}}{\textit{worst case } \texttt{collDetBSD}} &\approx \frac{2d^{(a+b)}}{(2d^a)(2d^b)} \\ &= \frac{d^{(a+b)}}{2d^a d^b} \\ &= \frac{1}{2}\end{aligned}$$

Now consider projection of the the second exaggerated omnimodel, Figure 5.10 (b), into the x' dimension. As illustrated in Figure 5.13, `collDetBSD` successfully classifies the whole region as *prohibited* on the first attempt. However,

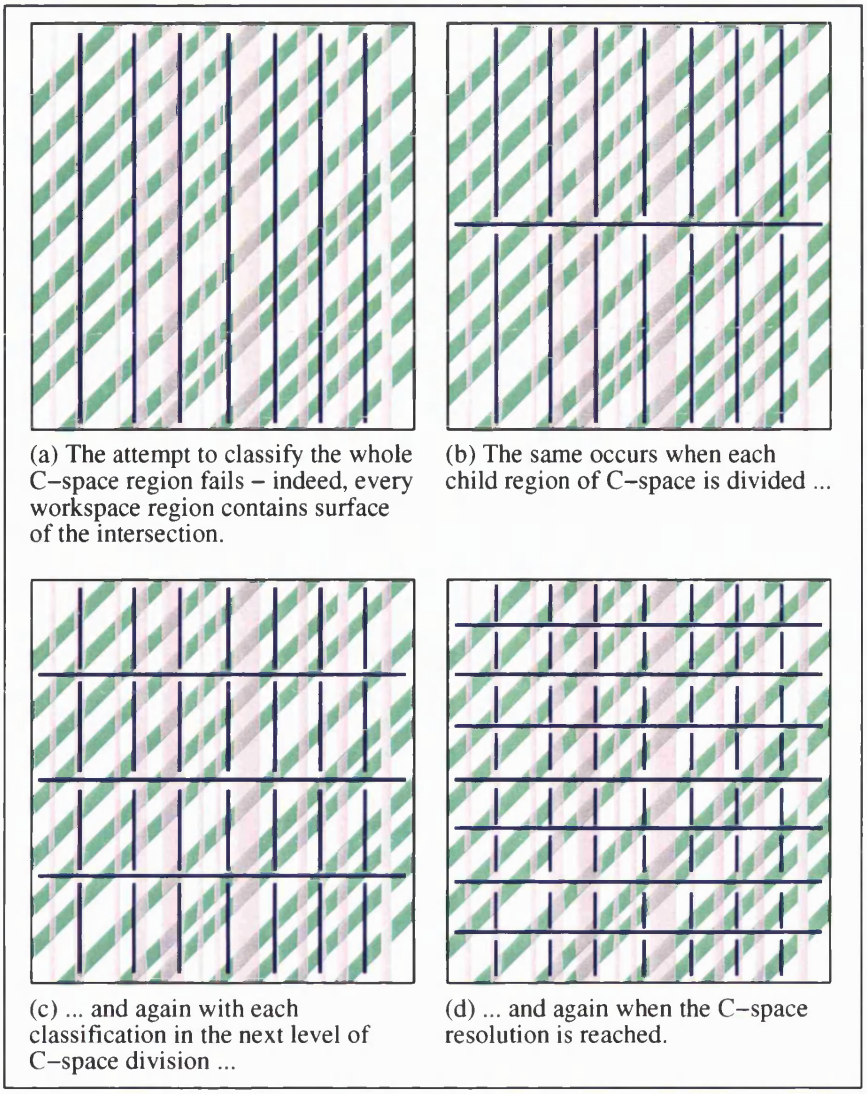
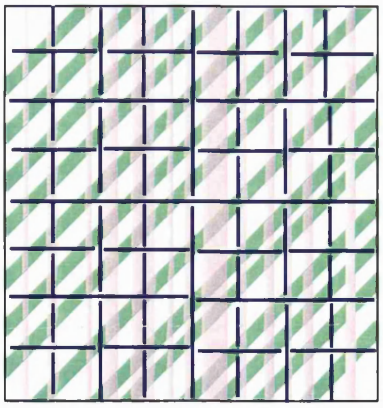


Figure 5.11: **An illustration of `collDetBSD` coarsely projecting a complicated omnimodel with fine detail**

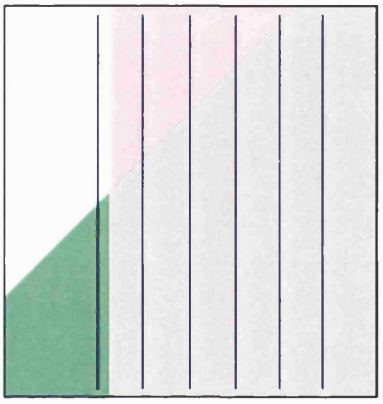
`isotropicBSD` (Figure 5.14) wastes effort adapting to the surface of the intersection region, only for cells classified later to reveal that the whole region is *prohibited.* Since the surface of the intersection may be multidimensional, mapping that surface with resolution-sized boxes can represent an enormous inefficiency.

In conclusion then, when the resolutions are coarse compared to the regions of intersection in the omnimodel, `isotropicBSD` is up to twice as fast as `collDetBSD`. However, when the omnimodel contains contiguous regions of intersection which are large with respect to the resolutions, `isotropicBSD` can be very inefficient due to its painstaking mapping of the surface of the intersection, which is unnecessary



Isotropic division never reaches a simple omnimodel, so the leaves are a regular grid.

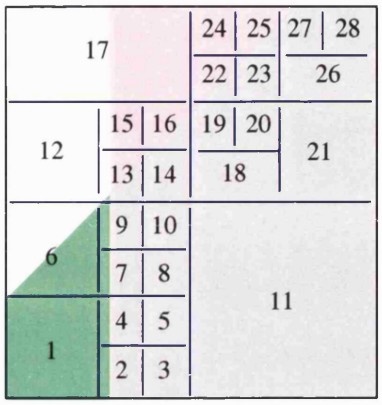
Figure 5.12: **An illustration of `isotropicBSD` coarsely projecting a complicated omnimodel with fine detail**



Division of the workspace across the complete C–space region reveals a solid box, indicating that the complete region is *prohibited*.

Figure 5.13: **An illustration of `collDetBSD` coarsely projecting a simple omnimodel containing a large intersection region**

for projection.



The leaves are numbered to indicate the order they are reached in the bottom–left–first division.

Note how division is wasted adapting to the edge of the intersection region, only for later cells (11, 21, 26, 28) to result in the whole region being classified as prohibited.

Figure 5.14: **An illustration of `istotropicBSD` coarsely projecting a simple omnimodel containing a large intersection region**

5.10 L-collDetBSD: 'Learning' collision detection for recursively sub-divided C-space boxes

5.10.1 Method

Remember that the weakness of `collDetBSD` was that effort was expended performing a full division of the workspace for C-space boxes which turned out to be too big to be successfully classified. `L-collDetBSD` is an improved version of `collDetBSD` which aims to take advantage of the 'unsuccessful' workspace division by handing on any information that was gained to help with classification of the child C-space regions.

Each time a region of C-space is classified, `L-collDetBSD` maintains a record of what parts of the workspace were found to be free from intersection for the parent C-space model; since both omnimodels created are concerned with subsets of the parent's C-space, regions which are known not to contain intersection for the parent *cannot* contain intersection for the children, so it is not necessary to repeat the workspace division in those areas. This is achieved as follows:

- As a preparation step (before division of the C-space commences) the complete omnimodel is recursively divided in the workspace dimensions, resulting in a model which I refer to as a `predivModel`. Each leaf in the `predivModel` is classified as `collision-free` (where interval arithmetic was able to rule out any possible interference between objects) or *possible-collision* (where division in the workspace dimensions terminated before such a classification could be made). Internal nodes are classified as *mixed* indicating that some part of the workspace represented is known to be collision-free.
- After the pre-division step, a C-space map is produced as before—by recursively chopping in half the region of C-space being considered, and classifying each region by dividing in the workspace dimensions. However, the classification stage benefits from the `predivModel` as follows:
  - If the current `predivModel` is classified as *collision-free*, no further division of that part of the workspace is required—it contributes *safe*

to the classification of the C-space box.

  - If the current `predivModel` is classified as *mixed*, don't bother pruning the omniset to the current part of the workspace; immediately divide the workspace and examine the two children. This action is taken since it is known that some of the box is collision free (for example half of it might be). Stepping immediately to smaller workspace boxes avoids expending energy on those already-classified parts of the workspace.
  - If the current `predivModel` is classified as *possible-collision*, prune the omniset to the current omnimodel to see if the last division in a C-space dimension has made it possible to classify the current C-space box. If a classification as *safe*, *prohibited* or *contact* is not possible, workspace division is required. However, instead of calculating a division plane then constructing two child workspace boxes, the two child boxes of the current `predivModel` are used. Each new omnimodel child is constructed by combining a child of `predivModel` with the current C-space intervals.

- Note that at each stage during the workspace division process, the current *predivModel* has the same workspace box as the current omnimodel being considered; each time the workspace is divided in half, the algorithm sends in the appropriate child of the `predivModel` as the current `predivModel`.

5.10.2 Strengths and weaknesses

The strengths of this scheme are that

- It maintains the ability of `collDetBSD` to spot large *prohibited* regions if the omnimodel contains intersections which are large compared to the resolutions.
- Wasted division in the workspace dimensions is reduced since as the motion of the nomad is limited, the amount of workspace which has to be examined shrinks.

However, this improvement does little or nothing to help when the resolutions are coarse compared to the intersections in the omnimodel.

5.11 Complexity analysis

We know that for pathological cases (like the one illustrated in Figure 5.10 (a) on page 107),

- All of the omnimodel projection algorithms examine $O(d^{a+b})$ cells, where d is the resolution in each dimension, a is the dimensionality of the workspace and b is the dimensionality of the C-space.
- `collDetBSD` and `L-collDetBSD` will examine twice as many cells as `collDetPoint`, `collDetBox` or `isotropicBSD`. `collDetPoint` and `collDetBox` examine less cells because they jump straight to the resolution-sized leaves without classifying nodes higher up the tree; `isotropicBSD` examines less cells because its division strategy is unfettered by full division in the workspace dimensions at each step.

We also know that as the C-space resolution becomes finer, the number of leaves in the C-space map produced by `collDetBSD`, `L-collDetBSD` and `isotropicBSD` tend towards being $O(d^{b-1})$ instead of $O(d^b)$, since the division adapts to the surface of the C-space obstacles. For the same reason, the total number of cells examined by `isotropicBSD` will tend towards $O(d^{a+b-1})$.

Beyond these assertions, complexity analysis is very difficult because of the heuristic nature of the algorithms and the large number of factors which affect the computational time and memory requirement. These factors can can be arranged into three groups:

5.11.1 The geometry of workspace objects

Factors include:

- The dimensionality of the workspace.
- The number of primitives used to define each object.

- The total number of linear halfspaces at the leaves.
- The maximum or average degrees of the surfaces.
- The volume, surface area or crinkliness of the objects.
- The orientation and location of the objects. (Adaptive division can be sensitive to both, since the depth of division depends upon the location of the object surface relative to the hyperplanes uses to divide the model.).
- The relative complexities of the obstacle and nomad sets. Is the complexity of the nomad a more significant factor than the complexity of the obstacle, or vice-versa, or is there no difference?
- The size and shape of the contiguous solid regions within the objects. Algorithms which terminate when a solid region in the omnimodel is found will perform more effectively if the objects have large contiguous solid regions.

5.11.2 The C-space map being produced

- The number of degrees of freedom (clearly this sets the dimensionality of the C-space and is a major contributor).
- The types of degrees of freedom (translational, rotation, or a combination). The types of degrees of freedom affect the shape of the swept omnimodel set and the number of halfspaces used to define it—these in turn affect the depth of division required and the time spent in pruning.

5.11.3 The resolution boxes

- The size and shape of the C-space resolution box.
- The size and shape of the workspace resolution.
- The relationship between the two.

5.12 Experimental results

5.12.1 Test platform

All of the C-space mapping algorithms described in this chapter were implemented in C++ using the svLis-m geometric modelling kernel. Except where stated, tests reported below were executed on a Silicon Graphics Onyx workstation, with two 150 MHz MIPS R4400 processors and 256 MB of memory. Results labelled 'Executed on an SG Origin' were executed on a Silicon Graphics Origin server with twenty 196 MHz MIPS R10000 processors and 6 GB of memory.

5.12.2 Performance criteria

Performance of the mapmaking algorithms is measured in terms of three criteria:

Percentage volume classified as *contact* The proportion of the result which is classified as *contact* is a measure of the uncertainty of the map.

CPU time The total amount of time spent executing in user mode, obtained using the Unix library routine `getrusage(3)`. Note that timings obtained in this way include operation system overhead, such as the time required to spawn a new process. Such overhead is not constant since it depends on operating system activity at the time, and thus errors are introduced into the experimental results. This error is particularly significant for very small execution times.

Max memory allocated The maximum resident set size utilised by the process, obtained using the library routine `getrusage(3)`. Again, this method of measurement introduces errors into the results since the operating system may allocate a larger block of memory to a process than it requires. Thus, the results define an upper limit on the memory requirement, and at low memory requirements this may be significantly larger than that actually required.

5.12.3 Resolution box properties

For the experiments reported, the resolution boxes had the following properties:

- The workspace resolution box has an aspect ratio of one. This is not always true of the workspace—indeed, since some sidelengths of the starting box may be more than twice as long as others, some dimensions may be divided to a greater depth than others.
- The resolution sidelength for translational degrees of freedom is equal to the sidelength of the workspace resolution box[1].
- The C-space resolution box has an aspect ratio of one in sides of the same type (for example, all sides which correspond to a translational degree of freedom are of equal length).
- Resolutions are measured in terms of the maximum depth of the division tree—that is, the maximum number of times a side of the original workspace or C-space box had to be chopped in half to make it smaller than the corresponding side of the resolution box.

5.12.4 Test results

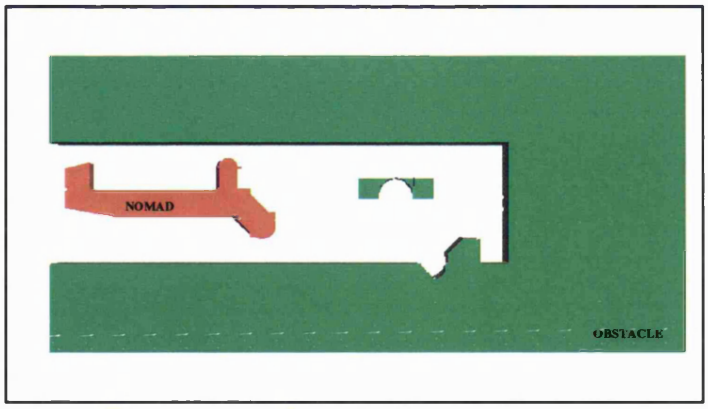
Clearly, the input criteria which affect the performance of the algorithms themselves define a highly dimensional space. Since the focus of this thesis is on breadth rather than depth, it was decided that instead of attempting an exhaustive exploration of this multidimensional space, a small set of cases would be tested to provide basic insights into the performance of the algorithms.

Some of the test cases are illustrated in Figures 5.15. Note that the tests predominantly used spherical obstacle and nomad sets. Although uninteresting in some senses, such cases have the advantages that the effect of orientation and location

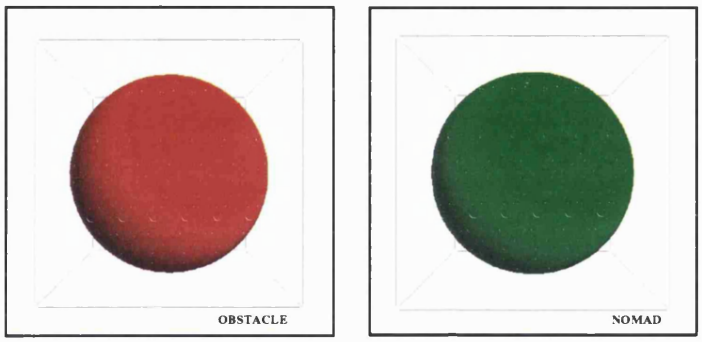
[1]In order to ensure every part of the nomad is able to travel to every part of the workspace, the translational box is typically obtained by computing the Minkowski difference of the obstacle and nomad workspace boxes. As a result, the sidelength of each translational side is typically twice as long as the corresponding workspace side—so the maximum depth of division in the C-space is typically one greater than the maximum depth of division in the workspace.

are minimised and that interactions within the multidimensional omnimodel can be imagined relatively easily.

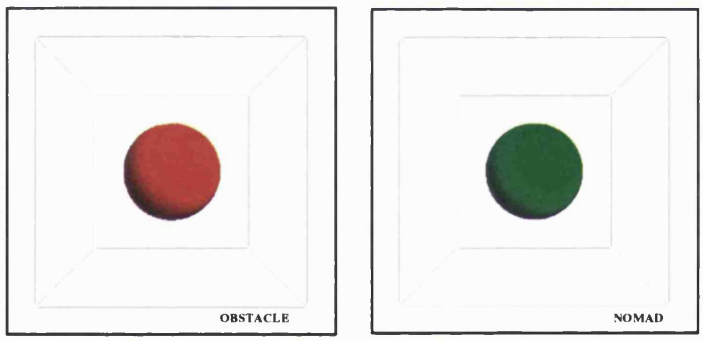
The findings can be summarised as follows.



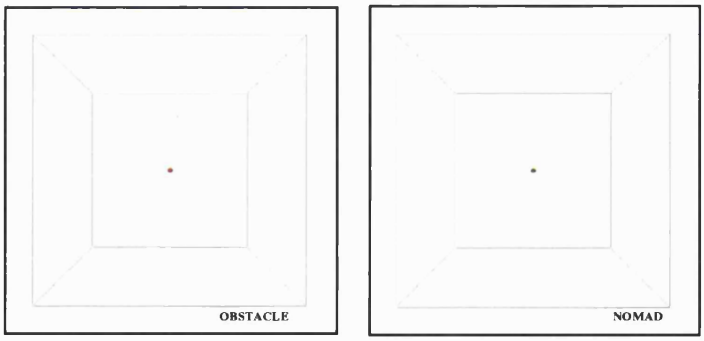
a) 'Gen–poly touch–latch' case



b) 'Big spheres' case



c) 'Medium spheres' case



d) 'Small spheres' case

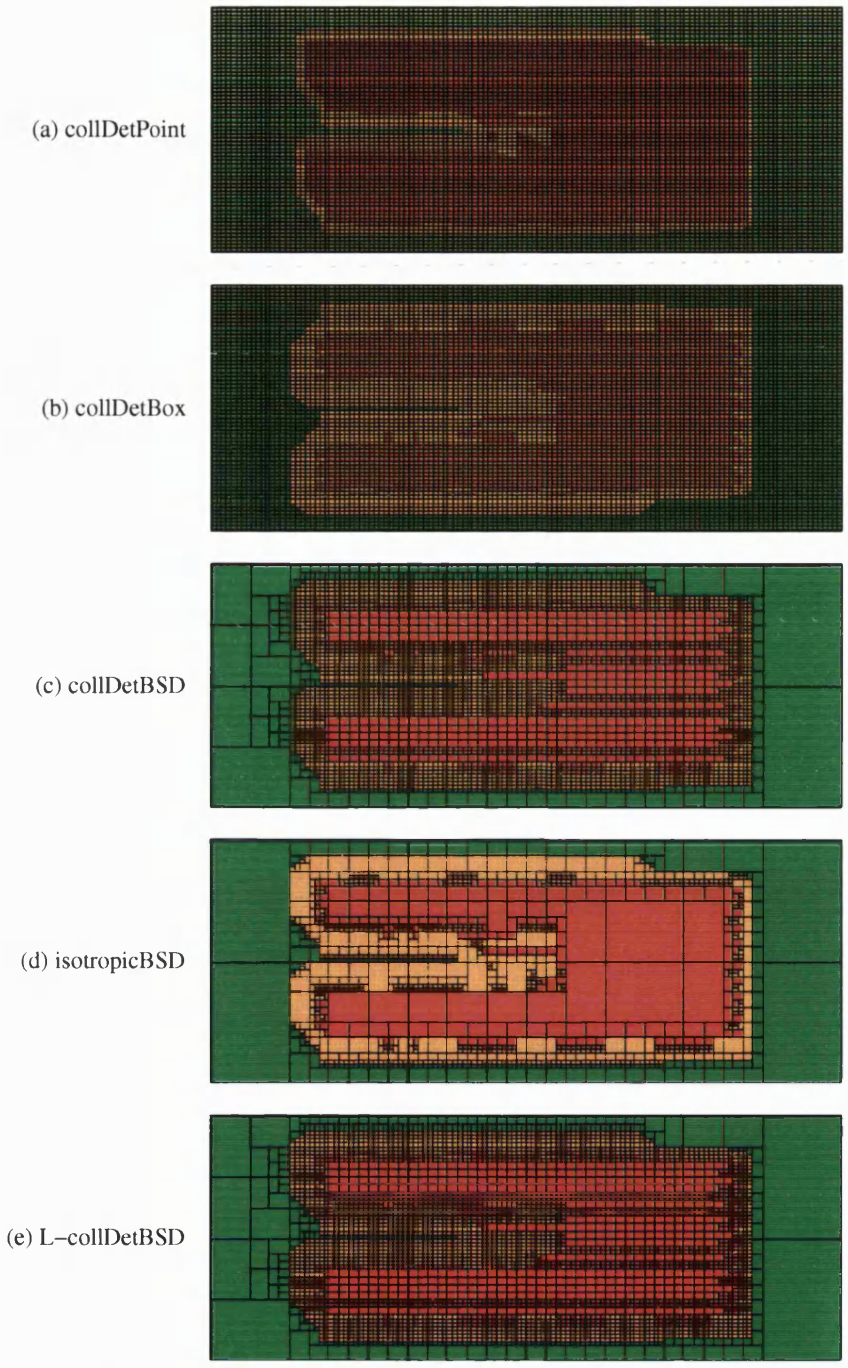
Figure 5.15: **Test cases referred to by graphs in this chapter**

Confirmation of correct behaviour

All of the algorithms compute an approximate map of the C-space correctly, as illustrated in Figure 5.16 for the 2-D 'Gen-poly touch-latch' case, which is a generalised-polytope approximation of two components of a touch-latch mechanism. The maps produced by `collDetBox`, `collDetBSD`, `L-collDetBSD` and `isotopicBSD` contain identical regions, whilst the maps produced `collDetPoint` have a thinner layer of *contact*[2].

In addition to translation-only C-space maps for a single nomad, the algorithms have been used to compute maps for two nomads with one translational degree of freedom each (Figure 5.17) and two nomads with one rotational degree of freedom each (Figure 5.18).

[2]Note that the maps produced by `isotopicBSD` appear to show a different division structure because of the additional merging which occurs—however, the maps are identical to those produced by `collDetBSD` and `L-collDetBSD` when cell-merging is enabled in those algorithms.



In each case, the max. depth of division in the C–space dimensions is 7 and the resolution in the workspace is the same as that in the C–space.

Figure 5.16: **A comparison of the maps computed by the five algorithms for the 2-D 'Gen-poly touch-latch' case with two translational degrees of freedom.**

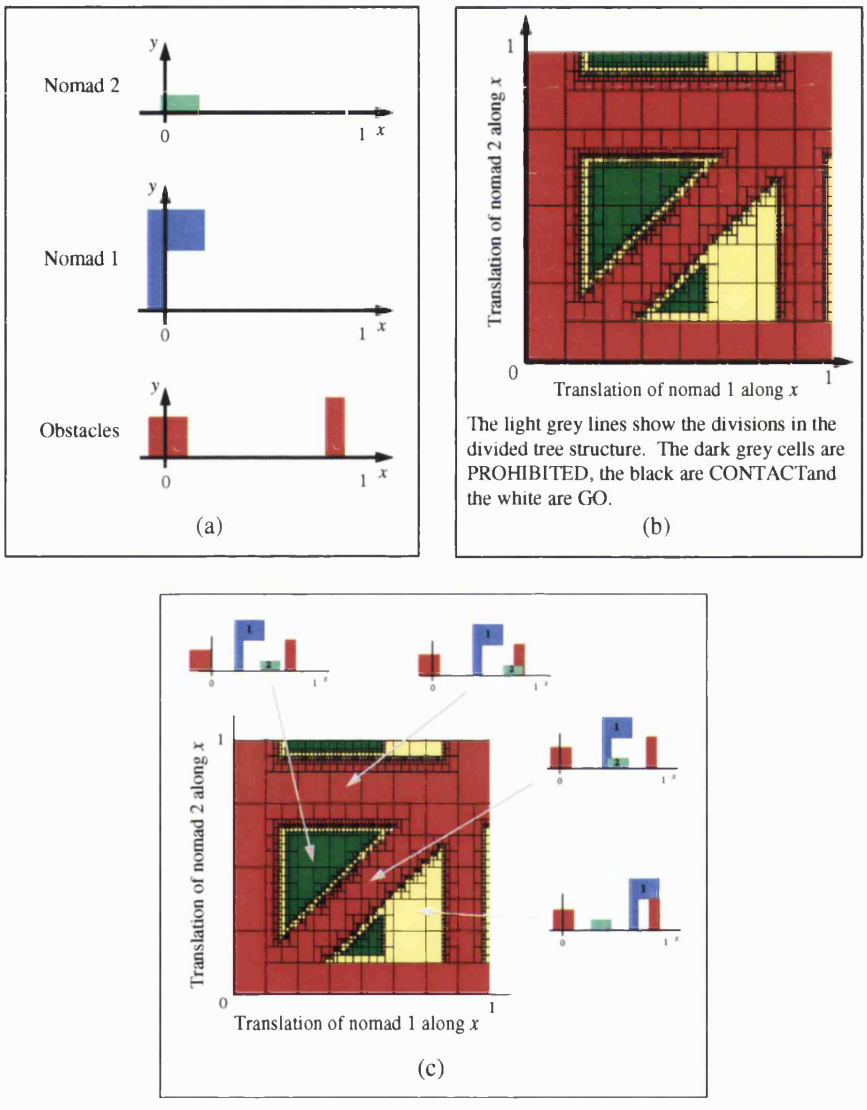
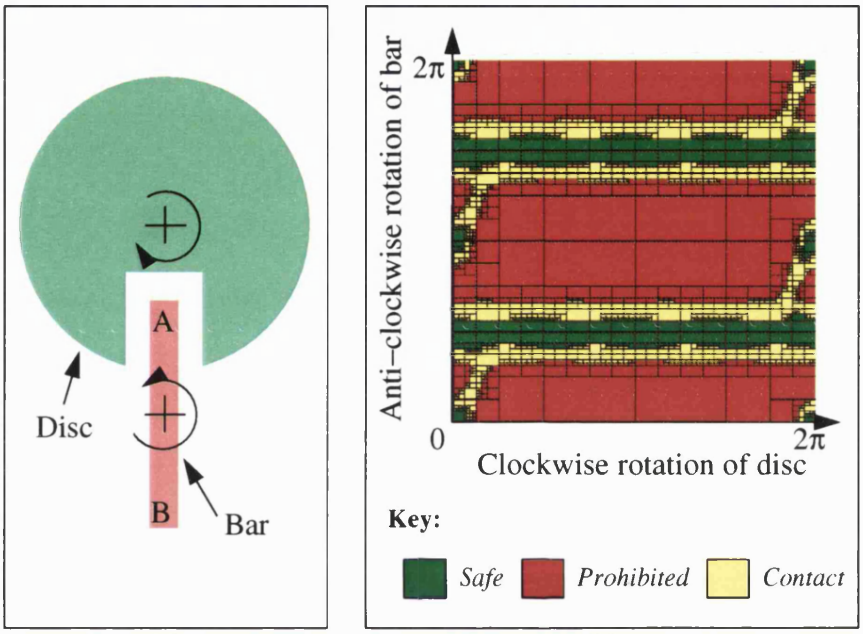


Figure 5.17: **Omnimodel projection has been used to compute the C-space map for a case with two nomads with one translational degree of freedom each.**



(a) An example two–dimensional mechanism – each object has one rotational degree of freedom.

(b) The C–space map for the mechanism in (a), computed using an omnimodel projection algorithm.

Figure 5.18: **Omnimodel projection has been used to compute the C-space map for a simple mechanism consisting of two components with one rotational degree of freedom each.**

The effect of C-space resolution on performance

Figure 5.19 shows the C-space map for the 2-D 'Gen-poly touch-latch' case with two translational degrees of freedom, at four different resolutions; Figures 5.20–5.22 plot percentage contact (by volume), time, and memory against division depth for the same case. Figures 5.23–5.26 show the equivalent graphs for the 2-D 'medium spheres' case[3].

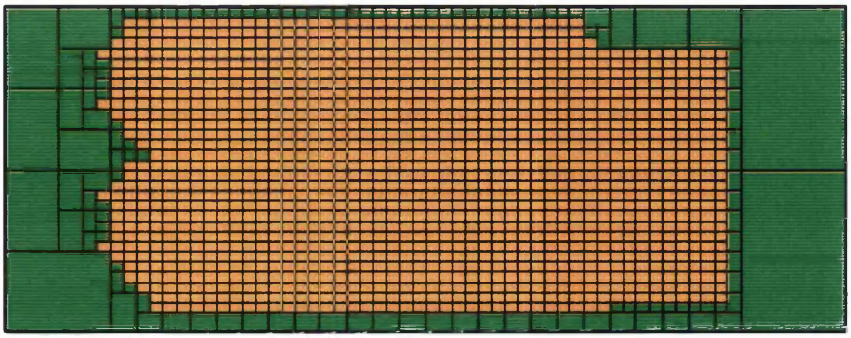
Notice firstly that for the touch latch case, the amount of contact falls slowly and erratically, as the resolution becomes fine enough to find solid boxes in increasingly narrow parts of the irregular intersection in the omnimodel. In contrast, for the 'medium spheres' case where the intersection is a more regular shape (akin to a skew four-dimensional cylinder) the percentage contact falls quickly in a fashion which a logarithmic scale confirms to be exponential (Figure 5.24).

In terms of time taken, different algorithms perform better in each case (Figures 5.21 and 5.28). For the 'medium spheres' case, the three algorithms which recursively divide the C-space perform better than the regular grid algorithms, which is to be expected since they are able to identify large regions of *safe* and *prohibited* early in the division. However, for the 'Gen-poly touch-latch' case, the thin sides of the obstacle prevent those algorithms from identifying *prohibited* cells until low in the division tree. As a result, the `collDetBSD` and `L-collDetBSD` algorithms, which waste division in the workspace dimensions looking for *prohibited*, are slower than the naive regular grid approach. The most significant property illustrated in these graphs, however, is that the linear curves on a logarithmic scale indicate that all of the algorithms take time exponential in the resolution. Only in benevolent cases do the more sophisticated algorithms gain more than an order of magnitude in time over the naive grid-based `collDetBox` (the time-resolution graph for such a case—'small spheres', where the C-space is almost entirely safe—is shown in Figure 5.27).

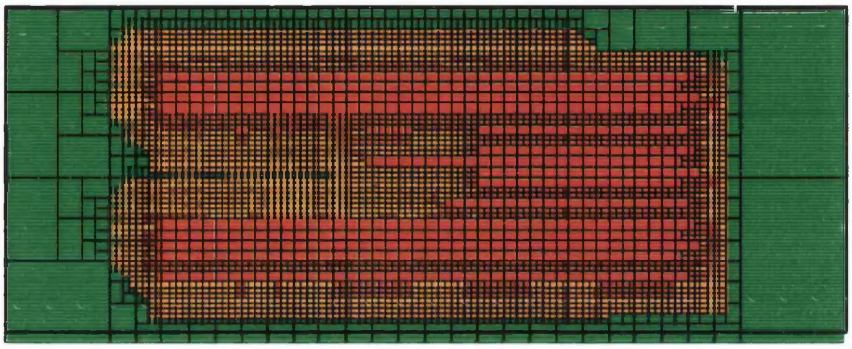
Regarding maximum memory usage, the pattern is the same for the two cases (Figures 5.22 and 5.26): memory requirement grows slowly for all algorithms except `L-collDetBSD`, which stores a record of the known map at each node. Note that in both cases, the memory usage curve is erratic for low depths of division, stabilising at a depth of six. These results appear to be due to the

[3]The data shown in Figure 5.25 is also provided in Table B.1 in Appendix B for reference.

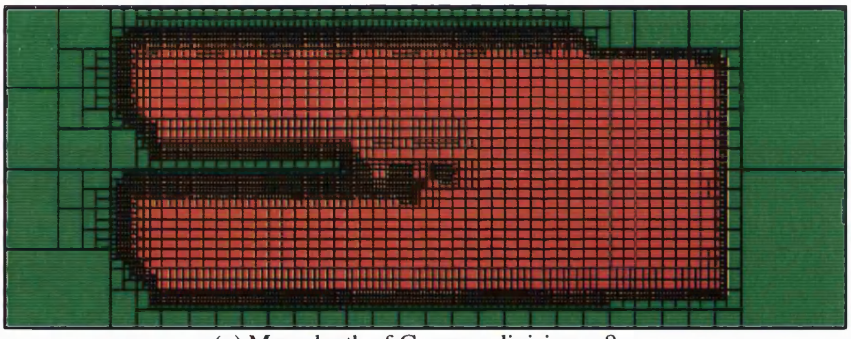
errors introduced by the method of measurement, as described in Section 5.12.2.



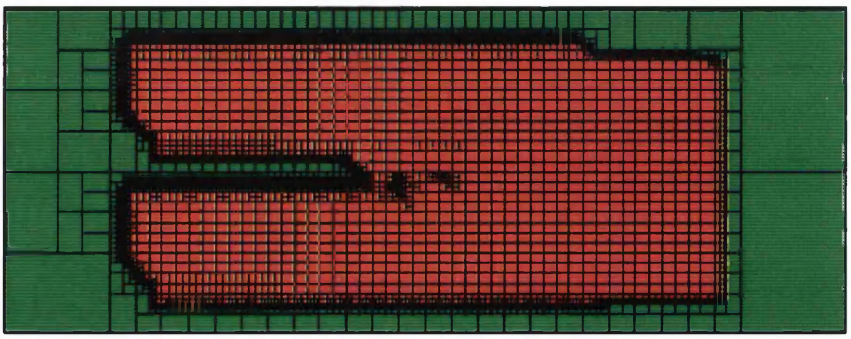
(a) Max. depth of C–space division = 6



(b) Max. depth of C–space division = 7



(c) Max. depth of C–space division = 8



(d) Max. depth of C–space division = 8

C–space maps computed by the `L-collDetBSD` algorithm for the 'Gen–poly touch–latch' case at different resolutions. At each level, the workspace resolution box is the same size as the C–space resolution box.

Figure 5.19: **C-space maps produced at different resolutions**

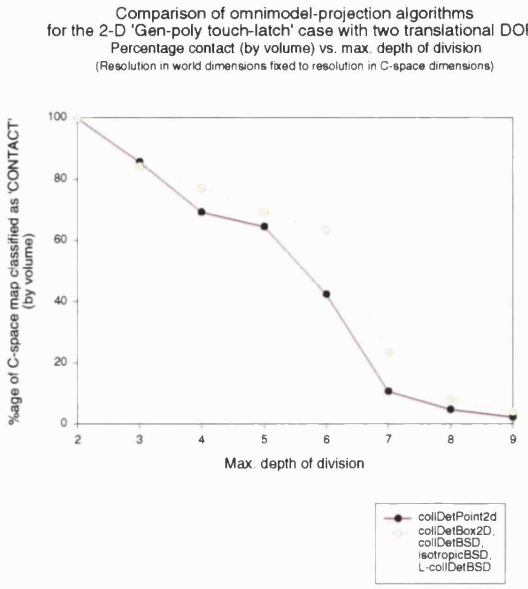


Figure 5.20: **Percentage CONTACT by volume vs. resolution for the 2-D 'Gen-poly touch-latch' case with 2 DOFs**

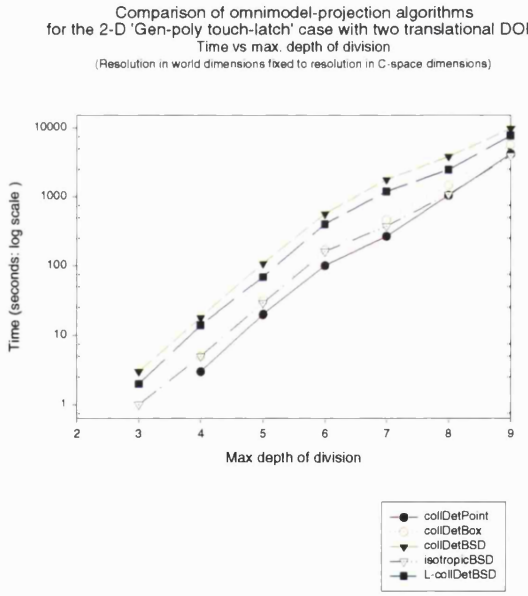


Figure 5.21: **Time taken vs. resolution for the 2-D 'Gen-poly touch-latch' case with 2 DOFs (note the scale is logarithmic)**

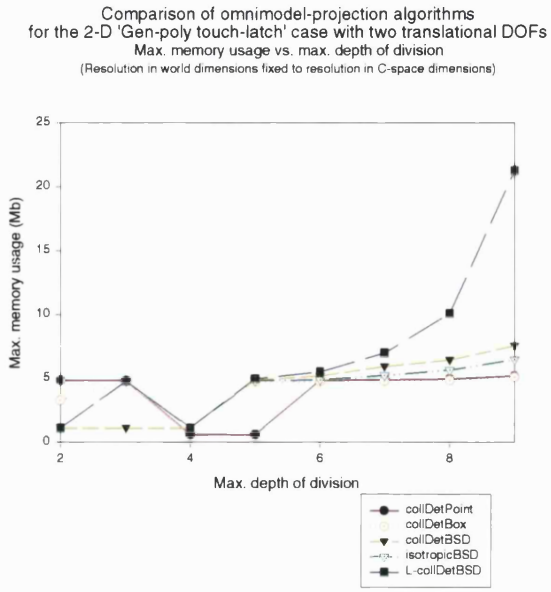


Figure 5.22: **Max. memory usage vs. resolution for the 2-D 'Gen-poly touch-latch' case with 2 DOFs**

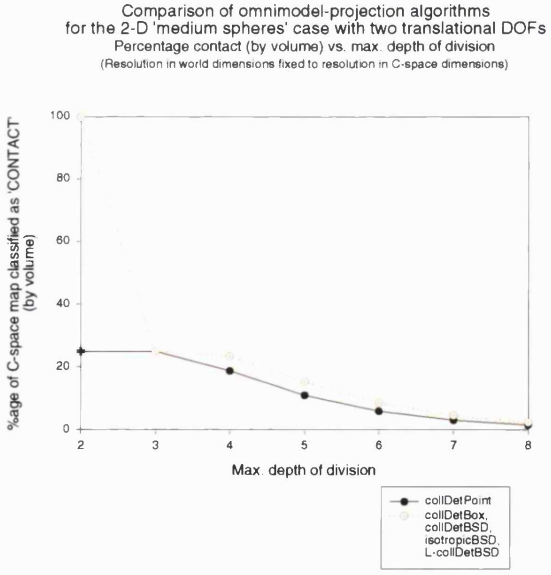


Figure 5.23: **Percentage CONTACT by volume vs. resolution for the 2-D 'Medium spheres' case with 2 DOFs**

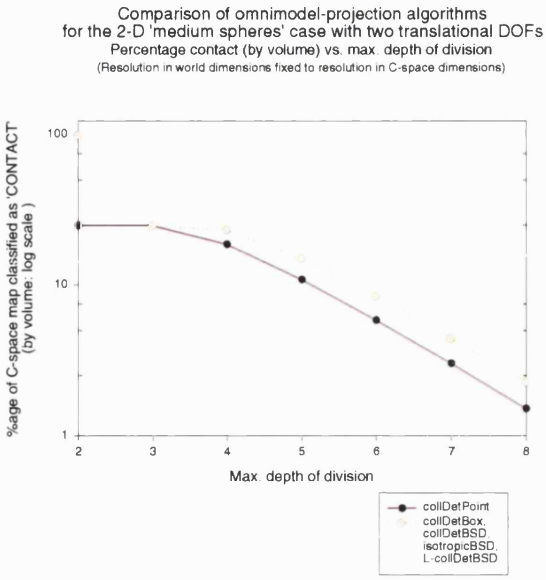


Figure 5.24: **Percentage CONTACT by volume vs. resolution for the 2-D 'Medium spheres' case with 2 DOFs (note the scale is logarithmic)**

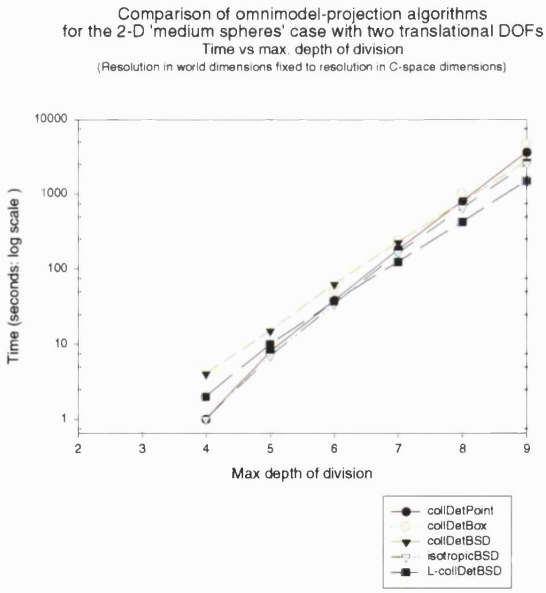


Figure 5.25: **Time taken vs. resolution for the 2-D 'Medium spheres' case with 2 DOFs (note the scale is logarithmic)**

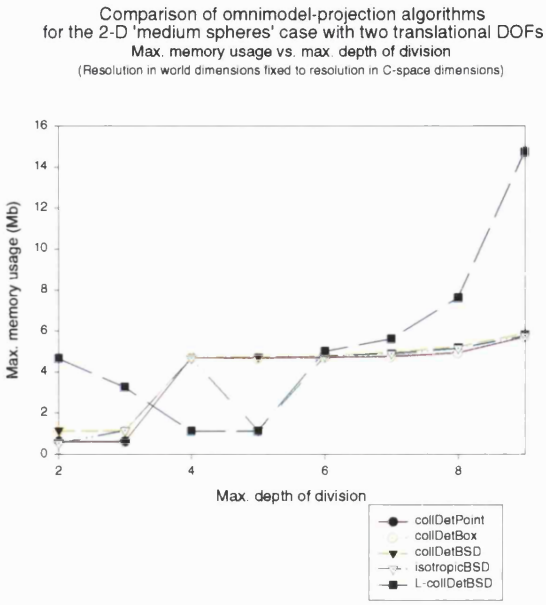


Figure 5.26: **Max. memory usage vs. resolution for the 2-D 'Medium spheres' case with 2 DOFs**

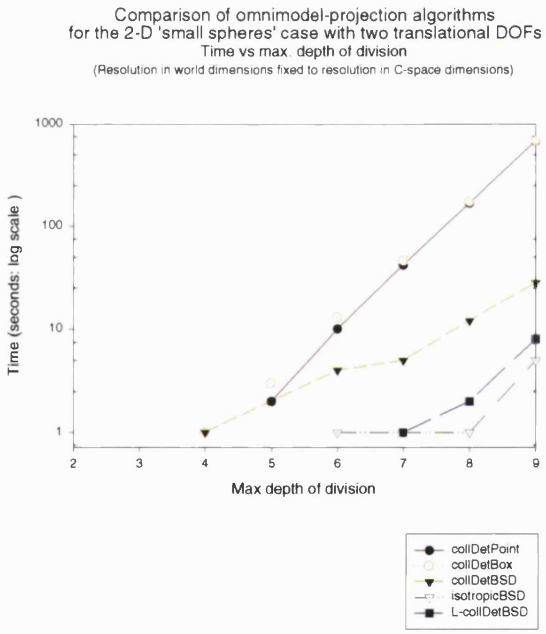


Figure 5.27: **Time vs. resolution for the 2-D 'Small spheres' case with 2 DOFs (note the logarithmic scale)**

The effect of the size of the workspace objects

As has been seen above, the shapes of the obstacles and nomad have a considerable effect on the performance of the algorithms. Figures 5.28 and 5.29 compare the time taken by each algorithm for the 'medium spheres' and 'big spheres' cases[4]. Although all the curves grow exponentially with resolution, the scales for these graphs are linear for clarity. Note that the increase in the sizes of the objects has two significant effects: firstly, most of the algorithms take approximately three times as long; secondly, `isotropicBSD` performs significantly less well—indeed, its speed advantage over `collDetPoint` is lost.

The latter effect can be accounted to the fact that `isotropicBSD` wastes time mapping the multidimensional surface of the interference region—which will be significantly larger for the larger case. The former effect is not surprising for the algorithms which recursively divide the C-space, since the surface area of the C-space obstacle (which determines the number of contact leaves in the result) is twice as large for the 'big spheres' case. What is surprising, however, is that the effect is almost as large on the regular grid algorithm. The significant increase in time, despite the constant number of tests, suggests that collision detection takes more time to identify *prohibited* cells than *safe* ones.

[4]The data shown in both these figures are also provided in tabular form in Appendix B for reference.

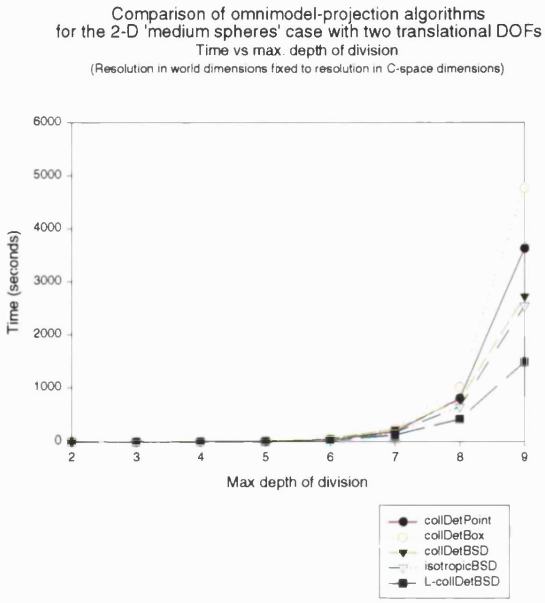


Figure 5.28: **Time vs. resolution for the 2-D 'Medium spheres' case with 2 DOFs**

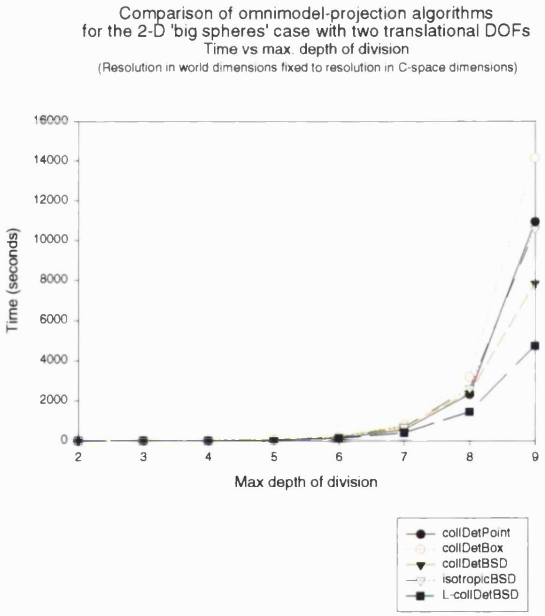


Figure 5.29: **Time vs. resolution for the 2-D 'Big spheres' case with 2 DOFs**

The effect of obstacle complexity

Figure 5.30 plots time taken against obstacle complexity for the three main algorithms (`collDetBox`, `isotropicBSD` and `L-collDetBSD`). The obstacle is a 2-D slice of a 'medium sphere', which has an increasing number of randomly placed tiny spheres subtracted from it, such that each one takes a tiny bit from its perimeter. Thus, its surface area and volume is approximately constant, whilst its complexity increases. Results are plotted for two nomads—'small sphere', which is so small that at the specified resolution no *prohibited* cells can be found, and 'medium sphere', which is the same size as the obstacle.

Note that in both cases for all algorithms, the time taken is linear in the number of primitives used to define the obstacle.

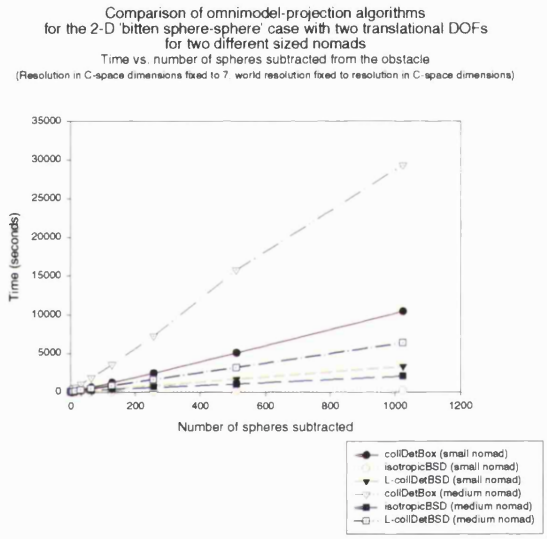


Figure 5.30: **Time vs. complexity of the obstacle for two sizes of nomad**

The effect of swapping obstacle and nomad sets

For translation-only C-space maps, the size and shape of the C-space obstacles are independent of which set is the obstacle and which is the nomad[5]. However, it is reasonable to expect the computation times to be affected by swapping the obstacle and nomad since, for example, the intersection region in the omnimodel between a tiny nomad and a large obstacle has different properties for projection than a large nomad with a tiny obstacle. Also, the omnimodel will require more halfspaces to define it if the nomad being swept is more complicated, since the transformation replaces each halfspace in the nomad with a tree (Section 5.3).

Figure 5.31 confirms that if the nomad and obstacle are of the same complexity but different sizes, swapping the sets makes a difference—but not a very significant one. `L-collDetBSD` and `isotropicBSD` (not shown) are slightly more effective when the nomad is bigger, which is to be expected since an intersection region which is thin in the workspace dimensions but wide across the C-space allows large *prohibited* regions to be identified sooner than the other way around.

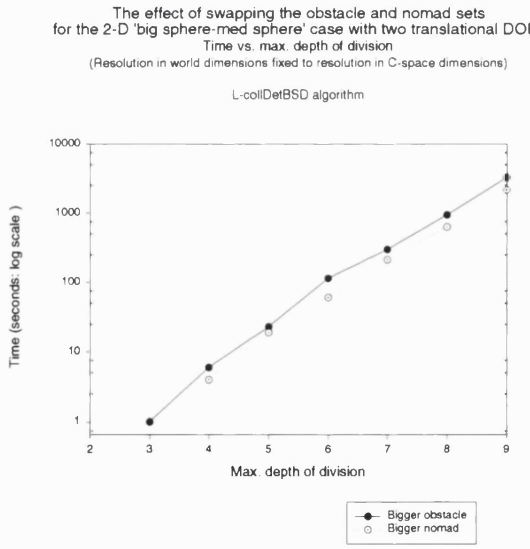


Figure 5.31: **The effect on `L-collDetBSD` of swapping the obstacle and nomad sets when one is significantly larger**

Figure 5.32 shows a more significant bias, indeed all of the algorithms which recursively divide the C-space are considerably less effective when the nomad is

[5]This is not true for C-spaces involving rotation—consider for example the case where one object is a sphere whilst the other is a long thin cuboid.

more complicated. As suggested above, this is due to the increased complexity of the omnimodel, which increases the cost of the pruning operations.

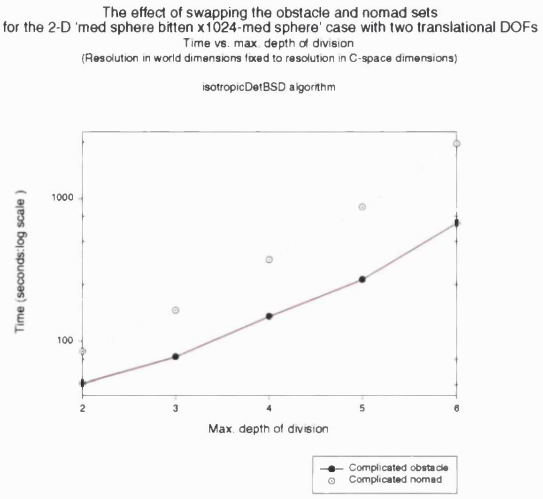


Figure 5.32: **The effect on `isotropicBSD` of swapping the obstacle and nomad sets when one is significantly more complicated**

The effect of workspace and C-space dimensionalities

Figures 5.33 and 5.34 compare the time vs. resolution curves for two- and three-dimensional versions of the 'Med spheres' case, both with two translational degrees of freedom, whilst Figure 5.35 introduces a third translation degree of freedom to the three-dimensional case[6].

These results lead us to the following observations[7]:

- The move from two to three dimensions does not favour `collDetBSD` and `L-collDetBSD` since their repeated division of the workspace is more expensive—indeed, `L-collDetBSD` becomes slower than the naive `collDetBox` algorithm.
- When the the number of degrees of freedom increases, adaptive division of C-space starts to pay dividends so the `L-collDetBSD` is faster than the regular grid approach.

However, these observations are restricted by the low depths of division studied—as illustrated in Figure 5.25, the more sophisticated algorithms gain advantage as higher division depths are reached.

The reason for the restriction to low division depths is, of course, the most significant limitation of the projection approach to computing C-space maps: since computation time is exponential in both the dimensionalities and the depth of division, computation times soon become impractical. As an example, the three-dimensional 'Med spheres' case with three translational degrees of freedom, the C-space map (which is a three-dimensional equivalent to the map shown in Figure 5.7 on p. 100) took approximately an hour and a quarter on an SG origin.

[6]The data shown in Figure 5.35 is also provided in tabular form in Appendix B for reference.

[7]Disregarding the apparently high times for very low depth of division in Figure 5.33, which appear to be due to errors introduced by the measurement process, as described in Section 5.12.2.

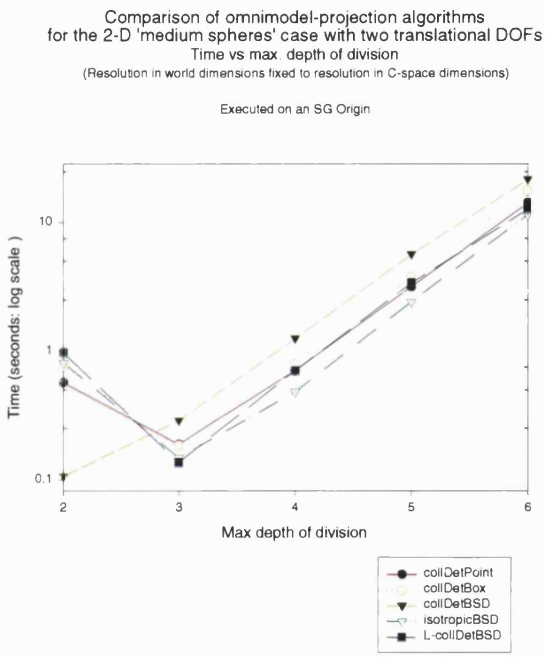


Figure 5.33: **Time (on an SG origin) vs. resolution for the 2-D 'Med spheres' case with two translational DOFs (note the logarithmic scale)**

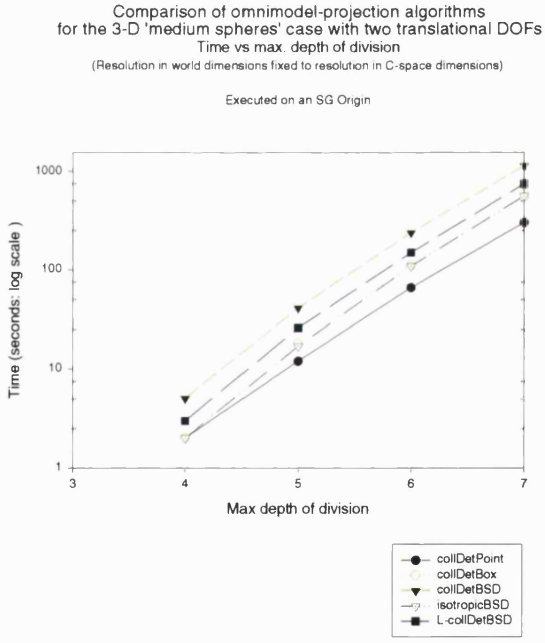


Figure 5.34: **Time (on an SG origin) vs. resolution for the 3-D 'Med spheres' case with two translational DOFs (note the logarithmic scale)**

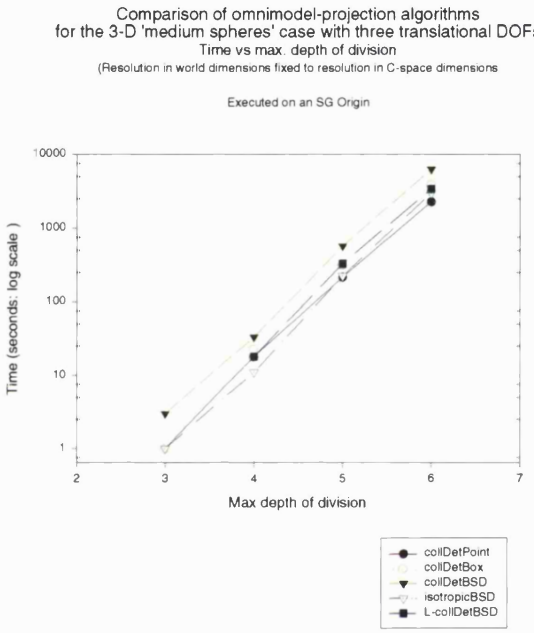


Figure 5.35: **Time (on an SG origin) vs. resolution for the 3-D 'Med spheres' case with three translational DOFs (note the logarithmic scale)**

5.13 Summary & conclusions

This chapter has demonstrated how multidimensional set-theoretic modelling can be used to construct a static model (referred to as an *omnimodel*) which represents every interaction that occurs as a nomad (or system of nomads) exercises its (or their) degrees of freedom. When the interference regions within this model are projected into the configuration space, they map the C-space obstacles to the nomad or system of nomads.

Five heuristic algorithms have been described which compute approximate C-space maps via projection in this way. All of these have been implemented, and some limited experimental results lead to the following conclusions:

- All of the algorithms are inherently general, and correctly compute maps for several different types of problem (a single moving nomad, multiple translational nomads, and multiple rotating nomads) and for workspaces of both two and three dimensions.
- Each of the five algorithms has its own strengths and weaknesses. However, disappointingly, despite the use of adaptive division and the introduction of some caching of information, none of the algorithms are more than an order of magnitude faster than the naive regular grid approach for the cases tested.
- The time complexity of all the algorithms is exponential in both the resolution and the dimensionality of the combined workspace and C-space. This makes the approach impractical (on its own) for cases where the combined workspace and C-space dimensionality is greater than four—at which dimensionality computation of a map with a resolution in each dimension of approximately 1% of the original length would take approximately 30 minutes on a contemporary PC.

However,

- The time and memory complexities of the algorithms are linear in the number of halfspaces defining the objects. This contrasts with the analytical

methods described in the next chapters, and suggests that the omnimodel framework might complement those methods by enabling complicated problems to be broken into simple subproblems which the analytical methods can solve very effectively.

Chapter 6

Computing precise C-space maps using set-theoretic modelling

6.1 Introduction

As described in the literature review (Section 2.4.2), Donald [24] derives predicates for contact surfaces and applicability conditions which he formulates in terms of Euler angles; he then demonstrates how the use of simplification and canonical forms enables trajectories to be intersected with the free-space. Using the same predicates, Canny [15] formulates contact surfaces using quaternions and is thus able to obtain a semi-algebraic expression for the free-space which he uses as the basis for a path planning algorithm using a Voronoi-style skeleton. Describing this approach, Latombe [53][p. 122] comments that the predicate method "does not provide an explicit representation of the boundary of the C-obstacle ... such as a list of faces, edges and vertices, with both their equations and topological adjacency relation.".

This chapter describes how, by exploiting properties of both the set-theoretic representation and Minkowski operations, I have implemented algorithms which, using the same approach as Donald and Canny, obtain precise analytical representations of C-space obstacles for a common group of cases. It is interesting to note that since the set-theoretic representation is used, it is not necessary to obtain an explicit representation of the boundary surface and both topological

and adjacency computations are avoided. It is also interesting to note that the algorithms which compute complicated multidimensional configuration space obstacles are derived from principles demonstrated in the solutions to very simple cases. For that reason, this chapter describes the simple algorithms in some depth then incrementally increases the complexity of the cases handled. Finally, note that

Throughout this chapter terms are used which were introduced in Section 2.3.1.

6.2 The Minkowski sum of closed convex polygons using set-theoretic modelling

In the general case, Minkowski sums are non-trivial to compute—for examples of work in the area see Bajaj and Kim [7], Ghosh [34], Kaul and Farouki [47] and Lee, Kim and Elber [57]. However, for the special case of two convex polygons, the situation is much simplified and there is a well known boundary representation solution (Lozano-Pérez [63]) which has a running time of $O(n+m)$, where m and n are the number of vertices of the convex polygons taken as input.

Now, since conversion between the boundary-representation and the set-theoretic representation is straightforward for such cases, the Minkowski sum of set-theoretically defined convex polygons *could* be obtained via B-rep and Lozano-Pérez' method. However, a concise algorithm can be obtained by exploiting the following properties of the set-theoretic representation and Minkowski sums:

- The Minkowski sum of two polytopes is itself a polytope.
- The Minkowski sum of two convex objects is itself convex.
- A convex polytope is equivalent to the intersection of a list of linear halfspaces.

The combination of these properties means that computing the Minkowski sum of two convex polygons corresponds to a mapping from two lists of intersected

linear halfspaces to a third list of intersected linear halfspaces. This in turn means that once a halfspace which forms an edge of the Minkowski sum is identified, that halfspace can be intersected with the rest—it is not necessary to compute vertices, adjacency information or even a set-theoretic tree.

Computing a Minkowski sum $OBS \oplus NOM$ is equivalent to calculating the shape swept out by NOM as its origin visits every point in OBS. From this analogy (and from examining an example such Figure 6.1) it is clear that some edges of $OBS \oplus NOM$ correspond to the path swept out by a vertex of NOM as its origin slides along an edge of OBS, whilst the other edges of $OBS \oplus NOM$ correspond to the imprint made by an edge of NOM as its origin reaches a vertex of OBS. In both cases, the halfspace required to form the edge corresponds to a halfspace from one of the input objects translated by a vector corresponding to a vertex of the other input object.

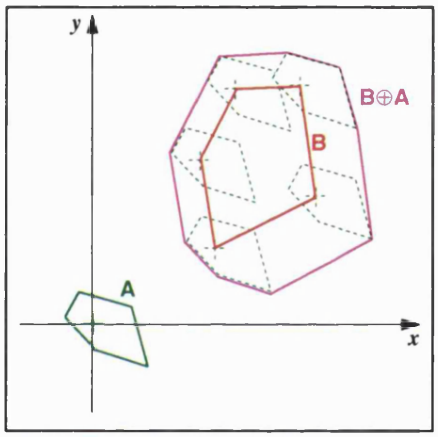


Figure 6.1: **An example Minkowski sum of two polygons**

If OBS is an m-gon and NOM is an n-gon, there are $2mn$ such halfspace-vertex combinations while $OBS \oplus NOM$ only has $(m + n - p)$ sides, where p is the number of halfspaces in OBS which have a coincident surface normal with a halfspace in NOM. Thus, the problem is identifying which of the $2mn$ combinations are used to define the Minkowski sum. In [63], Lozano-Pérez suggests two methods which solve the B-rep equivalent of this problem: one of these exploits an ordering of the vertices around each polygon and is thus only applicable to the two-dimensional case; the other generates every possibility then employs a convex hull computation. For *closed* polytopes, the set-theoretic representation allows a very simple solution:

1. For each of the n halfspaces of NOM

   (a) For each of the m vertices of OBS, translate the NOM halfspace by the OBS vertex.

   (b) Union together the m halfspaces which are generated in this way. Since they are all parallel (as illustrated in Figure 6.2) their union is equal to the one of them which is 'most solid'. Although all m halfspaces remain in the tree, pruning (Section 3.2.10) will remove the redundant halfspaces during evaluation.

2. Intersect the n sets obtained in this way to obtain a convex polygon, $RESULT_A$ (Figure 6.3 (a)).

3. Repeat the above process for each halfspace of OBS, translating each one by each vertex of NOM. Intersect the unions of the resulting halfspaces, to obtain a convex polygon $RESULT_B$ (Figure 6.3 (b)).

4. Intersect $RESULT_A$ and $RESULT_B$ to obtain the convex polygon $OBS \oplus NOM$ (Figure 6.3 (c)).

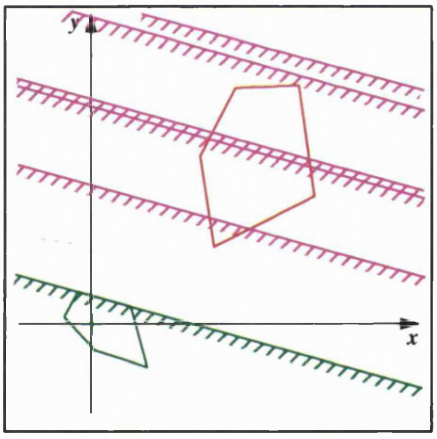


Figure 6.2: **One halfspace from NOM, translated by each of the vertices of OBS**

Note that instead of unioning together the different translations of a halfspace, the algorithm could identify which translated version is the 'most solid' and discard the others. This identification is trivial: the halfspaces each have the form $Ax + By + D \leq 0$ and since each instance is parallel, they will only differ in the D term; the one with the lowest D coefficient (which corresponds to the

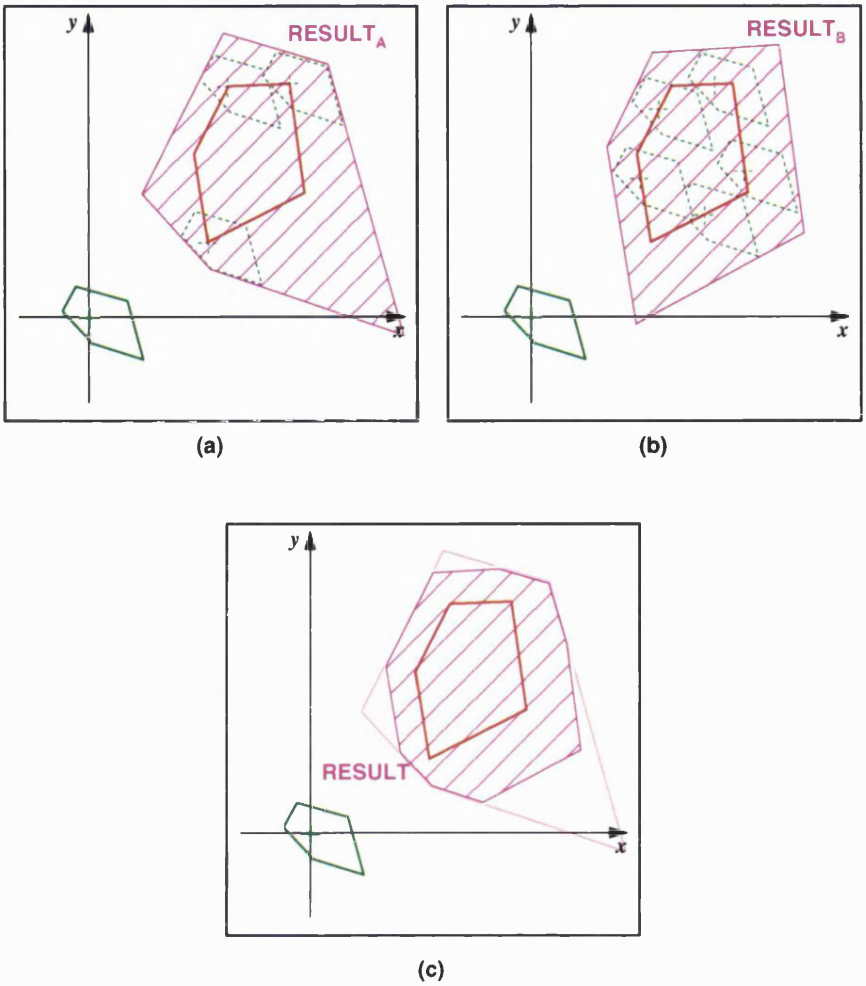


Figure 6.3: **Computing a Minkowski sum for a convex polygon**

signed distance of the origin relative to the surface) is the most solid. Performing this identification avoids both redundant halfspaces and coincident halfspaces (which occur if p (defined above) is non-zero). However, the unioning scheme avoids the comparison computation and has properties which can be exploited when rotational freedoms are introduced.

Regarding complexity, the unioning algorithm takes time $O(mn)$, which compares poorly to the $O(n+m)$ algorithm mentioned earlier. However, with extra code to handle the type C interactions, (Section 6.6) the unioning algorithm outlined above algorithm can be applied to the three dimensional case. Moreover, once rotations are introduced, the convexity of the objects ensures that every one of the $2mn$ contact conditions results in a contact surface which is applicable for some range of orientations. Thus, when the unioning algorithm is extended to

handle rotations, its complexity (which is still $O(mn)$) is optimal.

6.3 Computing C-space obstacles for translational cases of closed convex polygons

Section 2.3.1 described how, for a single nomad NOM which translates in a fixed orientation amidst obstacles OBS, obstacles in the configuration-space correspond to $OBS \ominus NOM \equiv OBS \oplus (\ominus NOM)$, where $(\ominus NOM)$ is NOM inverted (reflected in the origin). Now, if an object is defined set-theoretically using the linear halfspace basis, it is trivial to invert it—each leaf linear halfspace is replaced by its reflection in the origin and the set-theoretic expression remains unchanged. Thus, for translational cases of closed convex polygons, a precise analytical representation of the C-space obstacles can be obtained by inverting nomad in this way and then using the algorithm described in the previous section. This method was used by Cameron and Culley [14] to compute a C-space obstacle, enabling them to compute the minimum distance between two convex polyhedra.

Note that although configuration space constraints are embedded in a different space than the objects to which they correspond, the following sections, for simplicity, illustrate and compare the surfaces as if they were embedded in the same space.

6.4 Incorporating rotations

6.4.1 Face-vertex contact constraints

Figure 6.4 illustrates the construction of a contact surface corresponding to a Type A interaction, where a face of the nomad makes contact with a vertex of of the obstacle. Note that the contact surface has the same surface normal as the *reflected* nomad halfspace (c)—the surfaces are parallel and the prohibited side (shown hatched) corresponds to the solid side of the reflected halfspace (also shown hatched).

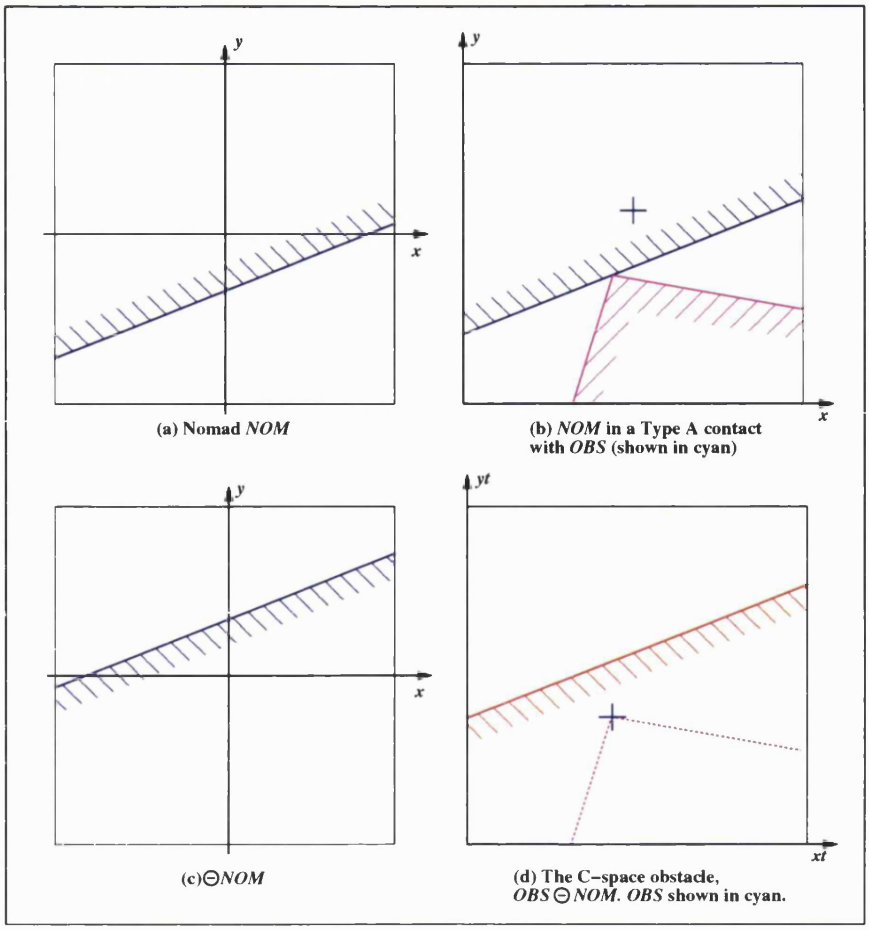
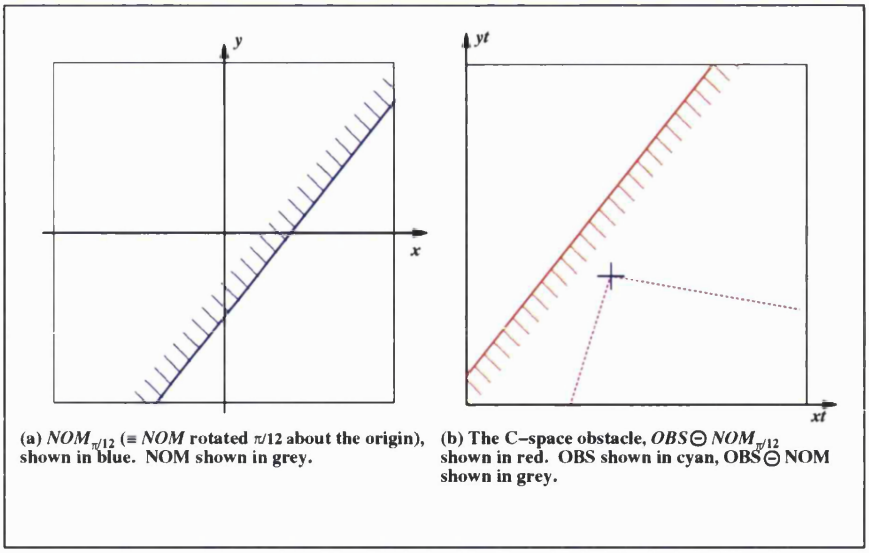


Figure 6.4: **Computing the constraint surface for a Type A interaction for a translation-only convex polygonal case**

When the nomad edge is rotated by some angle the C-space obstacle edge rotates by the same angle (Figure 6.5). Thus, when rotation in the plane is introduced to the convex polygonal case, each face-vertex contact results in a helical contact surface equal to the reflection of the nomad halfspace rotating around the obstacle vertex (as illustrated earlier in Figure 2.8 (a)).

An implicit representation of this helical halfspace is trivial and can be obtained using the algorithm which constructs an omnimodel with one rotational degree of freedom (see Section 5.3.2).



(a) $NOM_{\pi/12}$ (≡ *NOM* rotated π/12 about the origin), shown in blue. NOM shown in grey.

(b) The C–space obstacle, $OBS \ominus NOM_{\pi/12}$ shown in red. OBS shown in cyan, OBS⊖ NOM shown in grey.

Figure 6.5: **If the nomad is rotated, the face-vertex constraint surface rotates by the same amount**

6.4.2 Vertex-face contact constraints

Figure 6.6 illustrates the construction of a contact constraint corresponding to a Type B interaction, where a vertex of the nomad makes contact with a face of the obstacle. Note that in this case the contact constraint has the same surface normal as the obstacle halfspace—the surfaces are parallel, and the prohibited side corresponds to the solid side of the obstacle. The contact surface is offset from the obstacle by some signed distance V, where a positive value increases the solid region (as is the case in Figure 6.6 (d)) and a negative value decreases the solid region.

The value of V is equal to the signed distance (Section 3.2.4) of the nomad vertex v relative to the obstacle halfspace H when the nomad's reference point sits on the obstacle surface. Thus, using the offset property described in Section 3.2.4, the contact constraint can be obtained by subtracting V from the obstacle halfspace:

$$CS = H - V \tag{6.1}$$

V varies sinusoidally as the nomad rotates and is given by:

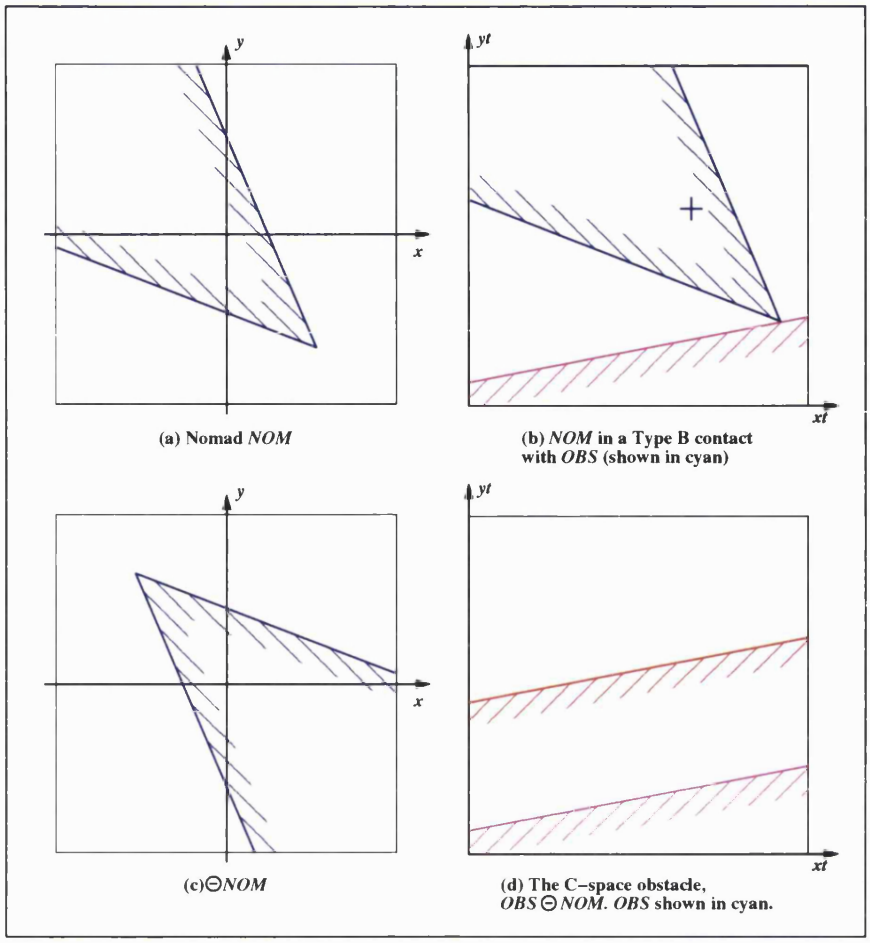


Figure 6.6: **Computing the constraint surface for a vertex-face interaction for a translation-only convex polygonal case**

$$V = \overline{v} * \cos(\theta + \theta_0) \tag{6.2}$$

Where $\overline{v}$ is the length of the line between the nomad's reference point and the vertex involved, and θ_0 is the angle that this line makes with the origin when the nomad is in its initial orientation.

An example of a sinusoidal constraint surface constructed in this way was illustrated earlier (Figure 2.8 (b)).

Equation 6.2 gives an appropriate surface for the two-dimensional case, but what would the equivalent be when the objects are three-dimensional and orientation is defined in terms of three parameters?

To derive an alternative solution which *can* be extended intuitively to all cases, let us remember from Equation 3.1 that the signed distance d, from a point p to a halfspace H is given by $H_A.p_x + H_B.p_y + H_D$, where H_{norm} is the surface normal of the halfspace and H_D is the signed distance of the origin relative to the halfspace. In this case, since the nomad's reference point is on the obstacle surface, H_D is zero. Thus, the signed distance of vertex v to the halfspace is given by

$$V = H_A.v_x + H_B.v_y \tag{6.3}$$

Substituting 6.3 in 6.1, each contact surface for a type B interaction is given by

$$CS_B = H - (H_A.v_x + H_B.v_y) \tag{6.4}$$

This is significant because if a rotation parameter θ is introduced, the x and y coordinates of vertex v can be obtained by expanding the rotation matrix to give

$$v_x = H_A \cos\theta - H_B \sin\theta v_y = H_A \sin\theta + H_B \cos\theta \tag{6.5}$$

Thus the implicit representation of a type B contact constraint can alternatively be obtained by substituting 6.5 into 6.1. This gives an unnecessarily complicated solution but, intuitively, this new solution can be extended to three dimensional rotation by simply replacing the two-dimensional rotation matrix by its three-dimensional equivalent.

6.4.3 Combining the contact constraints

For the case of a closed convex nomad which has full rotational freedom as it translates amidst closed convex obstacles, a precise representation of the C-space obstacles can be obtained by combining the contact constraints in exactly the same way as described in Section 6.2.

The closed convex nature of the objects ensures that *every* contact constraint is applicable for some range of rotations and ceases to be applicable when a contact constraint of another kind becomes applicable.

Unioning together all the contact constraints which involve a specific halfspace ensures that, for any orientation, the applicable constraint (which is the most solid of the constraints involving that halfspace) is correctly represented, whilst the others are redundant. As the rotation parameter varies through its range, the helicoid and sinusoid contact constraints each pass through each other, such that each one rises through the surface of the others when it becomes applicable, and sinks underneath for the remaining orientation. Type A contact constraints are bound by Type B contact constraints and vice-versa[1].

Figure 6.7 shows a C-space obstacle computed in this way for a case where the obstacle and nomad are rectangles.

6.5 Handling unbounded polygons

Consider the closed convex polygons shown in Figure 6.8 (a). The C-space obstacle which obstacle ABC causes to nomad DEF can be computed using the algorithm described in the previous section. Figure 6.8 (b) shows each of the edge-vertex contact surfaces which involve the nomad halfspace e, all sliced at a rotation value of zero. These are unioned together so that, for the rotation value shown, e-A and e-B are redundant, whilst e-C is intersected with other contact surfaces to form part of the complete C-space obstacle (Figure 6.8 (c)).

Now consider the same problem after halfspace b is removed from the obstacle, leaving it unbounded (Figure 6.9 (a)). Since the new obstacle has one vertex, only one Type A contact surface is generated (Figure 6.9 (b)). When this is intersected with the other contact surfaces it forms an erroneous contact surface, contributing to an incorrect representation of the C-space obstacle (Figure 6.9 (c)).

[1]Both Type A and Type B contact constraints can be bound by coincident contact constraints, which result when a nomad halfspace and an obstacle halfspace have coincident surface normals.

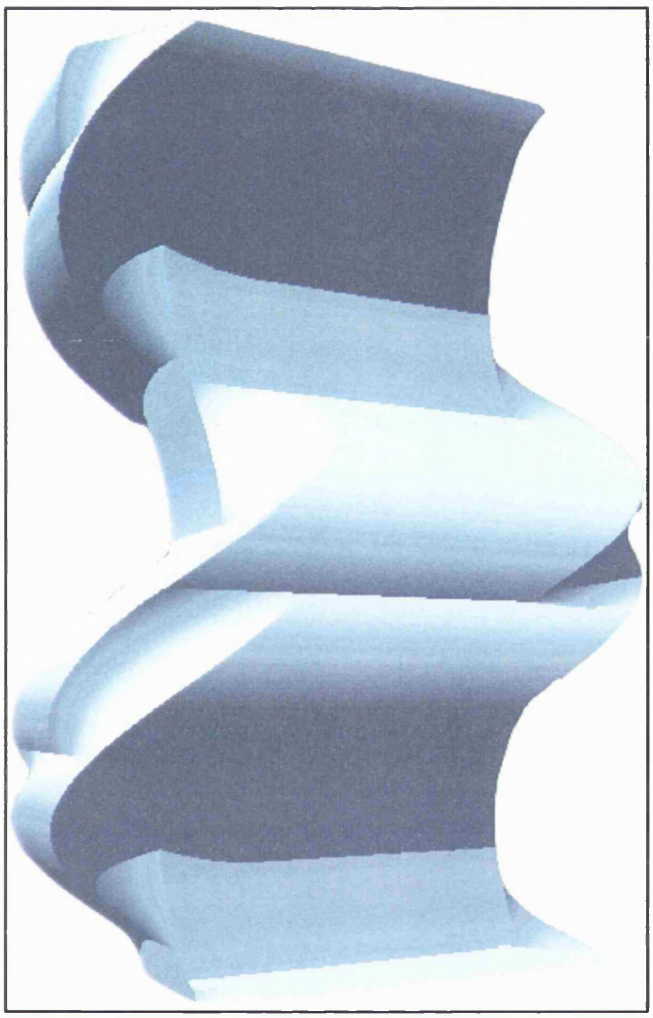
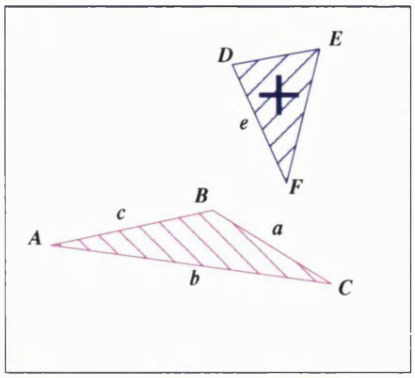


Figure 6.7: **A C-space obstacle for two rectangles. The horizontal plane represents the two translational degrees of freedom; the vertical axis represents rotation.**

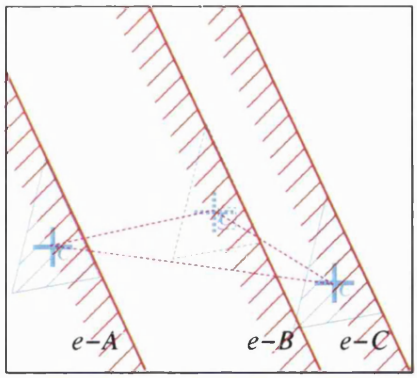
This problem is significant since although all objects in the real world are closed, unbounded objects will be very common if a problem is divided as part of a divide-and-conquer methodology, as discussed in Section 9.4.

The solution is to test every contact surface against an *applicability condition* such that only contacts which are applicable contribute to the C-space obstacle. This test can be formulated as a point-membership test as follows:

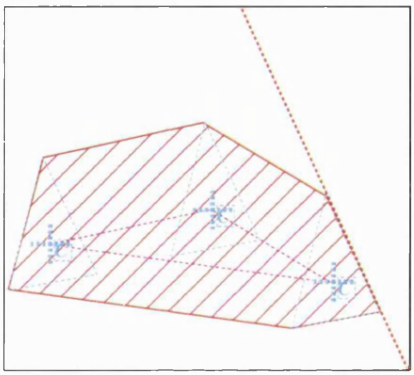
1. For each of the two halfspaces which meet at the vertex involved (v), create a halfspace which goes through the origin and has a surface normal pointing down the edge away from the vertex.



(a) Two closed convex polygons – obstacle ABC and nomad DEF



(b) The three edge–vertex contact surfaces involving edge e when the rotation parameter is zero.



(c) The complete C–space obstacle, sliced at a rotation parameter value of zero. e–C is shown dotted to illustrate its contribution.

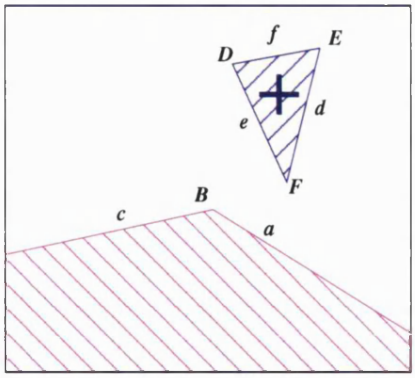
Figure 6.8: **An illustration of C-space obstacle computation for closed convex polygons**

2. Intersect the halfspaces created in this way to produce an *applicability cone.*

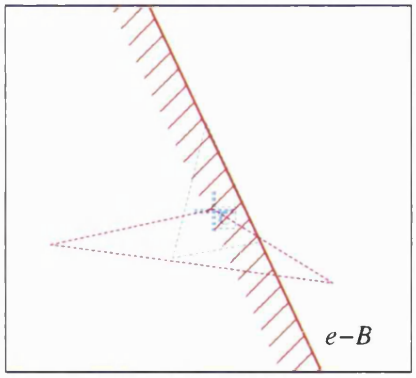
3. A contact involving v is applicable if and only if the surface normal of the edge involved lies within the applicability cone.

Figure 6.10 illustrates this algorithm for a vertex-edge interaction. Note that when the applicability condition is tested against in the extended Minkowski sum algorithm the *inverted* nomad model is used.

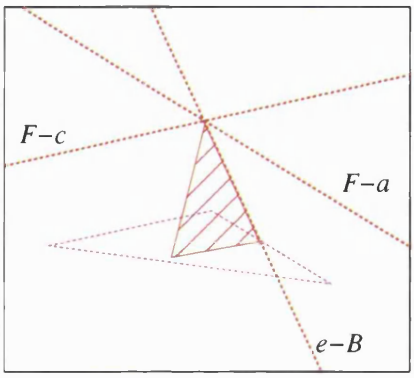
For translation-only cases, interactions which do not meet applicability condition



(a) The obstacle is made open when halfspace b is removed.



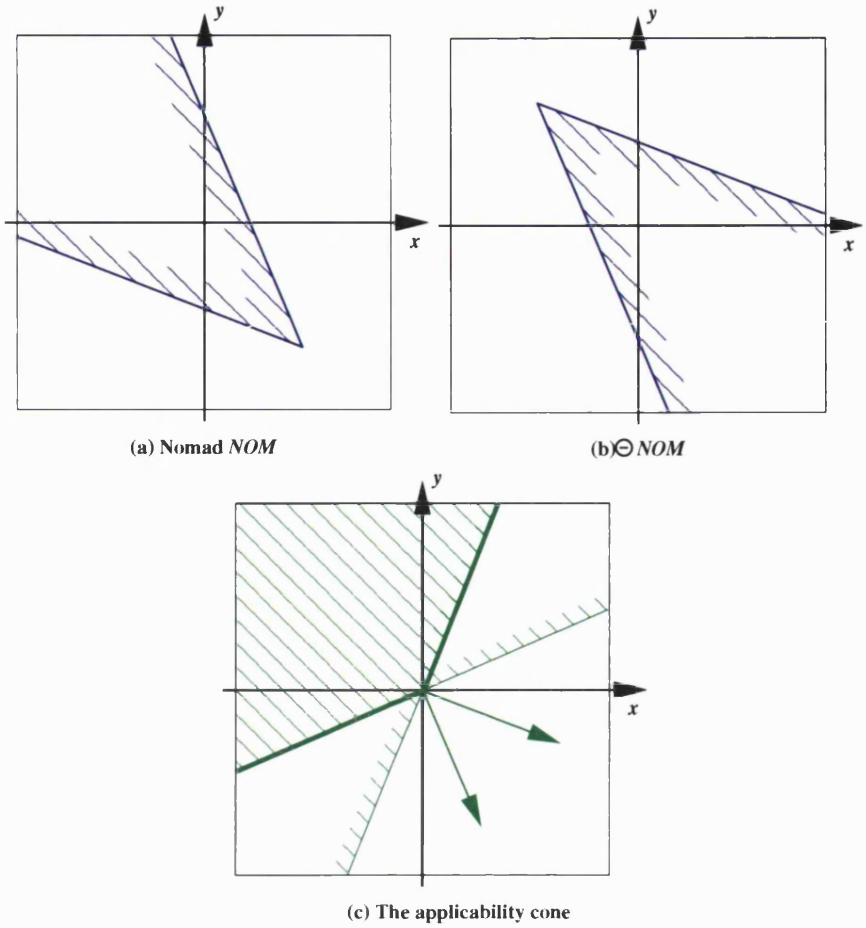
(b) Only one edge−vertex contact surface is generated.



(c) The erroneous C−space obstacle computed using the previous algorithm. Contact surface $e-B$ is shown dotted to illustrate its contribution, as are the vertex−edge contact surfaces $f-C$ and $f-a$, which are intersected with it.

Figure 6.9: **The same algorithm fails if one of the polygons is unbounded**

need not be introduced into the C-space obstacle representation. However, when rotation is introduced, each contact constraint is applicable for some range of orientations and not for the remainder. This is modelled by *intersecting each contact constraint against its applicability condition in the configuration space.* This cuts the contact constraint away when it is not applicable.



(a) Nomad *NOM* (b)⊖*NOM*

(c) The applicability cone

Figure 6.10: **Constructing an 'applicability cone' for a two-dimensional vertex-edge interaction**

6.5.1 Type A applicability conditions

In edge-vertex interactions, the vertex involved is stationary whilst the edge rotates. The applicability condition is represented as follows:

1. The surface normal of the nomad halfspace is multiplied by the appropriate rotation matrix to obtain the x and y components in terms of the (constant) initial values and the rotation parameter (see Section 5.3.2).

2. Membership testing a point p against a set S places the coordinates of p into the implicit halfspace equations at the leaves of S, combines the results according to the operators, and identifies if the value reached is less than or equal to zero—which indicates the point is inside the set. This operation

is formulated as a set by placing the expressions for the normal's x and y coordinates into the leaf halfspaces of the applicability cone. Since the only variable left in the tree is the rotation parameter, this set is embedded in configuration space; it is SOLID for the configurations in which the contact constraint is applicable, and AIR elsewhere

3. Each contact constraint is intersected with its applicability condition.

6.5.2 Type B applicability conditions

In vertex-edge interactions, the vertex involved rotates whilst the edge is stationary. The applicability condition is represented as follows:

1. The applicability cone is swept into the rotation dimension using the method described in Section 5.3.2.
2. The membership test is expanded in the way described for Type A applicability conditions—the coefficients of the obstacle halfspace normal (constants) are placed into the applicability cone to obtain a set in configuration space.
3. Each contact constraint is intersected with its applicability condition as before.

6.5.3 Handling cases where no interactions are applicable

Consider the case shown in Figure 6.11 (a) where the obstacle and nomad have both irregular shapes and polygonal segments, and the nomad is free to translate in a fixed orientation. A divide-and-conquer approach to solving this problem might reach the subproblem shown in Figure 6.11 (b)), focusing on the interaction between the polygonal segments of the objects. The unbounded objects of the subproblem are such that, in the fixed orientation of the nomad, no interactions are applicable.

What is the correct C-space map for subproblems like this?

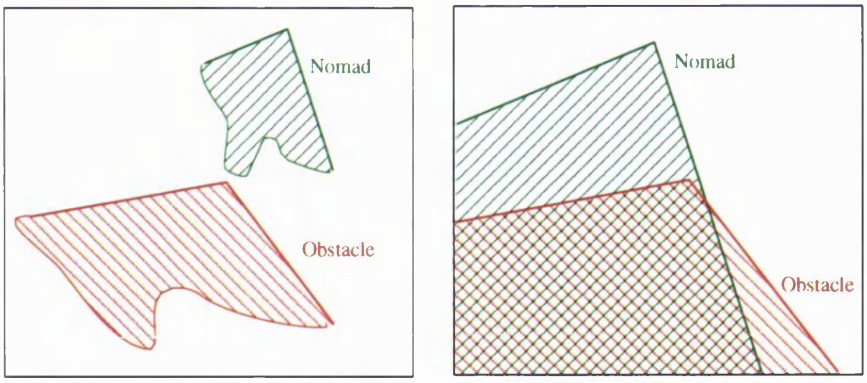


Figure 6.11: **A divide-and-conquer approach to problem (a) would lead to a subproblem (b) where no interaction is applicable.**

If each unbounded object as infinite, the correct result is a completely *prohibited* map. However, since one of my goals is to enable a divide-and-conquer methodology, and the framework for that methodology is the omnimodel division algorithm outlined in Chapter 5, the handling of unbounded objects must fit in with with that algorithm. Remember (page 97) that as soon one part of the workspace is found to contain a *prohibited* interaction, the recursion is terminated and the associated C-space box is classified as *prohibited*. Thus, if the subproblem of Figure 6.11 (a) is assigned a completely *prohibited* map, the whole problem (Figure 6.11 (b)) will be assigned a *prohibited* map—which is incorrect.

Thus, to fit in with omnimodel division, a subproblem which does not contain any applicable interactions should not contribute to the C-space map for the whole model—which can be achieved by returning a *safe* classification. The cost of this policy is that if a whole stand-alone problem contains an unbounded object, the completely *safe* C-space map which results is incorrect.

A simple solution which meets these conflicting demands is to assume that all cases being analysed start out as bounded, and that unbounded objects only occur as sub-problems. This assumption can be safely made if all input objects are forced to be bounded by intersecting them with a cuboid set constructed out of their model's box.

6.6 From polygons to polyhedra

The dimension-independent characteristic of the set-theoretic representation means that the only significant extension required for the algorithms described above to handle polyhedra is that contact constraints and applicability conditions must be formulated for Type C (edge-edge) interactions.

6.6.1 Type C interactions for translation-only cases

As with Types A and B, all Type C contact constraints are ruled surfaces—for any fixed orientation the surface is a linear halfspace. The surface normal,n of this halfspace is perpendicular to both edges which cause the constraint, so it is given either by:

$$n = e_1 \times e_2 \tag{6.6}$$

or its opposite

$$n = -(e_1 \times e_2) \tag{6.7}$$

Since the contact constraint has prohibited and safe sides, and the edge vectors give no indication as to which side should be which; either direction could be the correct surface normal.

Also, since edge-edge constraints are not generated in groups of parallel surfaces, the unioning scheme described for Types A and B is not appropriate—each contact surface must be tested for applicability even for the case of closed convex polyhedra.

These two complications are handled as follows:

For each edge-edge interaction,

1. The surface normal obtained from the cross product is tested for applicability against each edge.

2. If the surfaces normal meets the applicability condition for both edges, the normal is correct and the constraint is applicable.

3. If the constraint fails to meet either of the conditions then a second constraint is generated, with the normal in the opposite direction. If this second constraint meets both the applicability conditions it is correct and applicable—otherwise neither constraint is applicable.

4. When a correct and applicable surface normal is identified, all that is required to compute the contact constraint is a point on the surface. From the definition of Minkowski sum, every point on the contact constraint surface is the sum of a point on one edge and a point on the other edge, so such a point can be obtained by adding arbitrary points from each of the edges.

5. Testing if a contact constraint is applicable to an edge can be reduced to testing if a two-dimensional Type A or Type B constraint is applicable. This is achieved by setting up a local coordinate system where the third axis is parallel to the edge so that the two halfspace forming the edge project into lines which form a vertex, and the contact constraint halfspace projects into a line. The applicability cone algorithms described earlier can therefore be used.

6.6.2 Type C interactions incorporating rotations

Rotations are incorporated into the Type C contact constraints as follows:

1. Each nomad edge vector is multiplied by the rotation matrix (p.89) to obtain each coordinate in terms of the rotation parameters, as described earlier.

2. The cross product is formulated as a set-tree representing its expanded form. The cross product, c of a and b is given by:

$$
\begin{aligned}
c_x &= a_y b_z - b_y a_z \\
c_y &= b_x a_z - a_x b_z \\
c_z &= a_x y_b - b_x a_y
\end{aligned} \tag{6.8}
$$

3. The obstacle edge vector coordinates (constants) and the nomad edge vector coordinates (set-trees) are substituted into the leaves of the expanded cross-product to obtain a set for each coordinate of a vector which is perpendicular to both edges as the nomad rotates.

4. Testing each applicability condition corresponds to membership testing the normal vector against an applicability cone. As before, this is formulated by

   (a) Making applicability cones for nomad edges rotate by using the expanded rotation matrix.

   (b) Expanding the membership test operation into a set-tree, and placing the expressions for the coordinates of the normal into the leaves. This generates an applicability condition in terms of a set in configuration-space.

   (c) Intersecting each contact constraint with its applicability condition.

6.6.3 Type A and Type B interactions

Type A and Type B interactions (which are now face-vertex and vertex-face, respectively) are handled identically to the two-dimensional case except that:

- All halfspaces take the form $Ax + By + Cz + D \leq 0$.
- The rotation matrices used to compute the contact constraints and applicability conditions are expanded to incorporate the z dimension and any additional rotation parameters.
- Each applicability cone is defined by as many halfspaces as meet at the vertex—which can be any number greater than or equal to three.

Figure 2.7 (page 38) shows a configuration-space obstacle (for a translation-only problem) computed using this method.

6.7 Handling non-convex polytopes

As stated at the outset, the algorithm built up over the previous sections is only applicable to cases where both polytopes are convex. However, Minkowski sums are distributive over union:

$$A \oplus (B \cup C) \equiv (A \oplus B) \cup (A \oplus C) \tag{6.9}$$

Therefore, cases of non-convex polytopes can be handled by breaking each object into convex pieces, solving each combination and unioning the results.

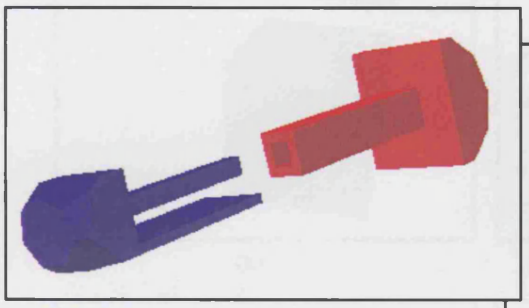
The general problem of convex decomposition is beyond the scope of this thesis. However, a simple solution is to rewrite each set in *disjunctive form* which represents an object as the union of a finite number of possibly-overlapping *products*, each of which is the intersection of primitives (in the polytope case, all linear halfspaces). This is achieved by recursively applying this identity to the set-tree:

$$A \cap (B \cup C) \equiv (A \cap B) \cup (A \cap C) \tag{6.10}$$

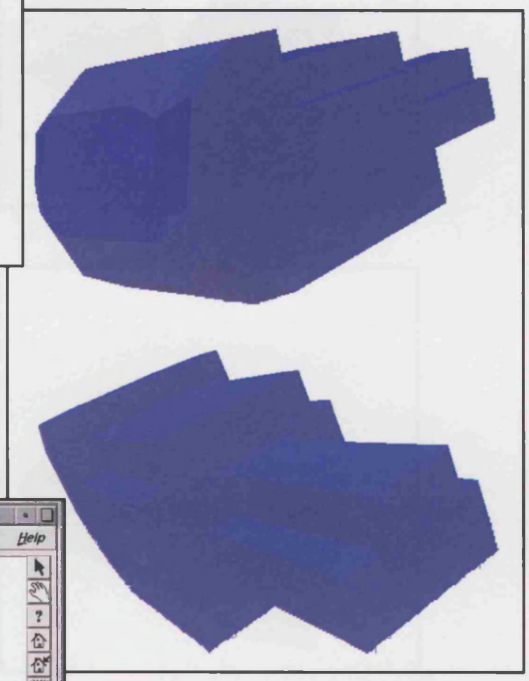
In the worst case, the number of products generated by this simple decomposition grows exponentially: if the original set-tree contains $2K$ linear halfspaces and has intersection operators everywhere except the K lowest internal nodes, the disjunctive form will contain 2^K product terms, each of which is the intersection of K halfspaces (Rossignac [84]). However, it is practical for many interesting cases, so it is sufficient for our purposes. Moreover, the algorithms described here are independent of the method used to obtain a convex decomposition, making it easy to plug-in a more efficient decomposition algorithm.

Figure 6.12 shows a configuration-space obstacle computed using this method for

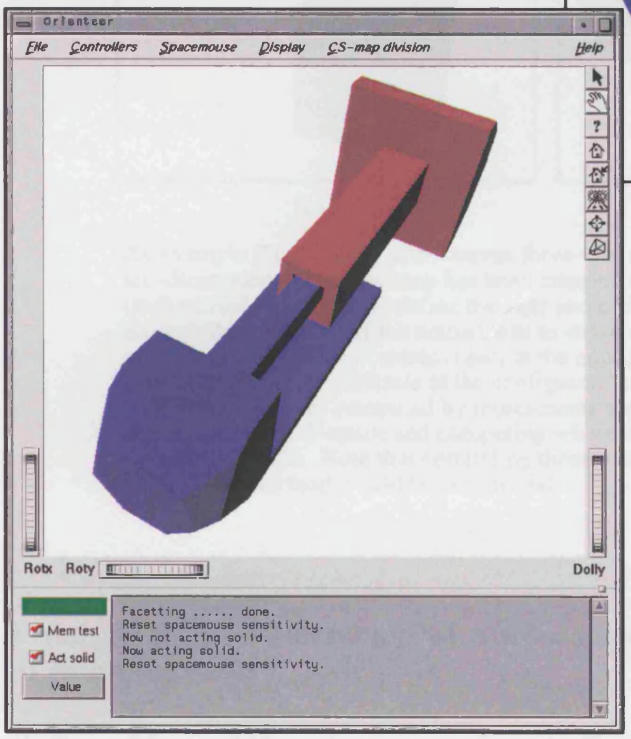
a translation-only case involving non-convex polyhedra; Figure 6.13 illustrates a non-convex polyhedral case for which a six-dimensional C-space map has been computed.



(a) Two non–convex polyhedral components

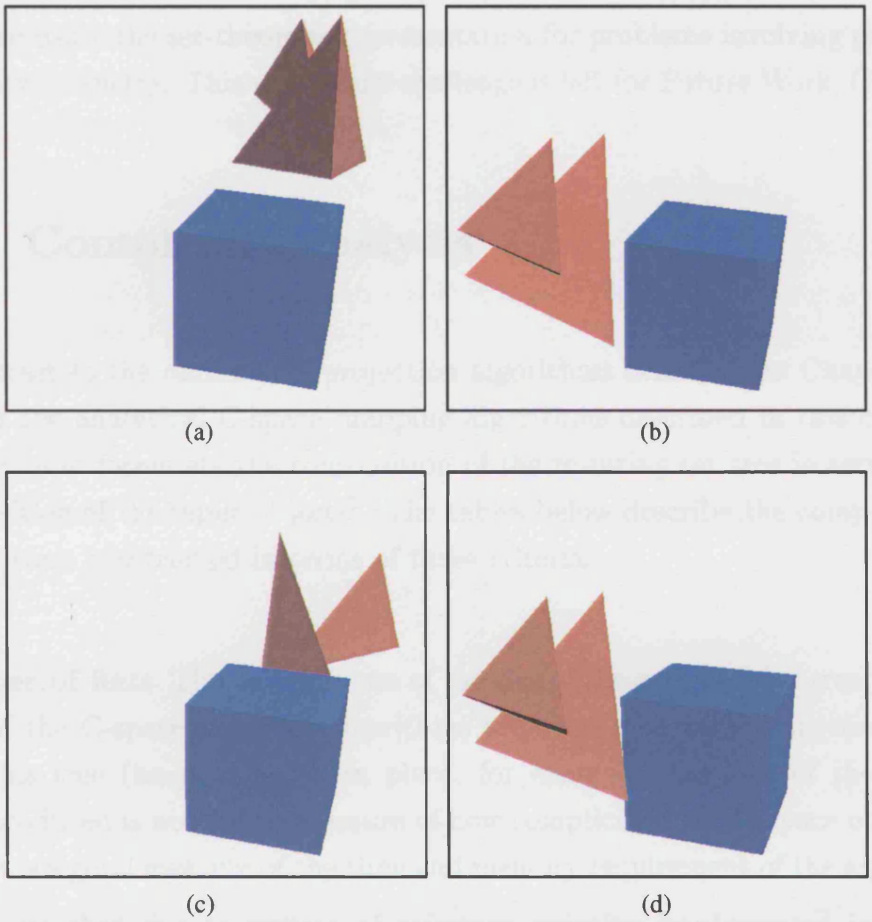


(b) Two views of the polyhedral C–space obstacle when one component is free to translate relative to the other in a fixed orientation



(c) Orienteer is used to validate the C–space map by checking the objects behave correctly. It confirms the map is correct, and also demonstates that mating between the parts is just possible.

Figure 6.12: **An illustration of a non-convex 3-D translational case**



An example illustrating a non–convex three–dimensional case for which a precise six–dimensional C–space map has been computed. Let the configurations of the (red) nomad in (a) and (b) define the start and end configurations of a linear path through the C–space. If the nomad was to move along that path, it would first make contact with the obstacle (blue) at the configuration shown in (c) and would stop overlapping the obstacle at the configuration shown in (d). These configurations were computed by representing the path as a ray through the six–dimensional C–space and computing where that ray entered and exited the C–space obstacle. Note that computing these configurations without computing a global C–space map would be non–trivial.

Figure 6.13: **An illustration of a non-convex 3-D case with six degrees of freedom**

6.8 Handling curved surfaces

Latombe [53, p. 149] proposes that if obstacle A and nomad B are represented as semi-algebraic sets, the C-space obstacle that A causes to B can be formulated as a Tarski sentence—it is therefore a semi-algebraic subset of the configuration space. This suggests that, in theory, a precise configuration-space map can be

obtained using the set-theoretic representation for problems involving practically arbitrary geometry. This significant challenge is left for Future Work, Chapter 9.

6.9 Complexity analysis

In contrast to the omnimodel projection algorithms described in Chapter 5, for each of the analytical C-space mapping algorithms described in this chapter it is possible to formulate the composition of the resulting set tree in terms of the composition of the input objects[2]. The tables below describe the composition of the set-trees constructed in terms of three criteria:

Number of flats This is a measure of the size of the complete set tree. Since all of the C-space mapping algorithms are concerned *only* with constructing this tree (no pruning takes place, for example) the size of the set tree produced is not only a measure of how complicated the C-space obstacle is, it is a good measure of the time and memory requirement of the algorithm[3].

Note that due to system of reference pointing implemented in svLis-m the number of flats stored in memory will be considerably less than this number—a large proportion of the flats in the tree (implemented as a graph) will merely be pointers to other nodes.

Note also that the numbers shown in the table are upper limits—some of the algorithms give simpler results if the input objects contain axially-aligned planes. This stems from the fact that svLis-m treats an axially-aligned plane as a one-dimensional set regardless of the space it is embedded in (Section 3.4.4).

Number of primitives This is a count of the number of *differentiable sets* (sets which do not contain boolean operations and are the equivalent of svLis primitives). The larger this number is as a proportion of the number of

[2]For the three-dimensional translation-only case this is not quite true—only *applicable* type C interactions will be introduced into the result set, the number of which will depend on the shape and orientation of the input objects. The tables provided quote the number of Type C contact surfaces *considered*, which in any case more accurately reflects the time complexity of the algorithms.

[3]The only case where the size of the result tree does not reflect the amount of work carried out is the three-dimensional translation only case, where effort is expended considering all edge-edge interactions but only applicable ones are introduced to the result

flats, the more effective pruning will be, since recursively subdividing a model containing one very complicated primitive will have no effect, whilst division of a set containing many simple primitives will result in a number of simpler models.

Maximum primitive contents This is the highest number of flats within a single primitive in the result. This is a measure how complicated the most complicated surface type is, which will affect the accuracy of interval arithmetic if the C-space map is recursively divided.

Analysis of the analytical C-space mapping algorithms is provided in the following tables:

Table 6.1 - composition of each type of contact constraint Note that for Type C contact constraints, the set tree required to represent the applicability cone is also incorporated since that applicability cone is required even when the input objects are known to be bounded.

Table 6.2 - composition of a Type A or B applicability cone Note that the size of the applicability cone will depend upon the number of edges meeting at the vertex involved (k, say).

Table 6.3 - composition of a complete C-space obstacle These results are for a convex nomad with n faces and a convex obstacle with m faces.

Note that for the three-dimensional case, it is assumed that every vertex has three edges meeting at it ($k = 3$). This enables the number of edges (e) and the number of vertices (v) to be formulated in terms of the number of faces (f), via the Euler-Poincare formula for an object which does not contain holes:

$$f + v - e = 2 \tag{6.11}$$

Therefore,

$$e = 3(f - 2) \tag{6.12}$$

and

$$v = 2(f - 2) \tag{6.13}$$

Note also that if both input objects are known to be bounded, the C-space obstacle for cases involving rotation can be defined using fewer flats, since applicability conditions are not required for Type A and Type B contact conditions.

Table 6.4 Example set-tree results for two specific input cases—rectangles in two dimensions ($n = m = 4$) and cuboids in three dimensions ($n = m = 6$).

From these tables, the following observations can be made:

- Type C interactions are by far the most expensive to compute and represent. Indeed, closer examination reveals that in the cuboidal case where the nomad has a full six degrees of freedom, approximately 98% of the C-space obstacle's set tree is used to represent edge-edge interactions.
- For the three-dimensional case, moving from one rotational degree of freedom to three rotational degrees of freedom does not change the number of primitives (since the same number of constraints and applicability cones are required). However, since all of these become more complicated, the number of flats increases by a factor of approximately four.
- Although unbounded cases require applicability cones to be introduced for Type A and Type B contacts, these are small compared to set required to represent Type C contacts, so the set tree only increases by about 6%.
- A three-dimensional case with a full six degrees of freedom results in a set-tree approximately four hundred times bigger than the two-dimensional equivalent with a full three degrees of freedom.

Note that although some of the values in Table 6.4 for cases involving rotation may seem large, they are optimal in that none of primitives which define the contact constraints or the applicability conditions are redundant. A more efficient representation could only be achieved by representing each primitive more efficiently, for example by using an alternative parameterisation of rotation such as quaternions.

Constraint type	*Dimen -sions*	*DOFs*		*Input*		*Output*		
		Trans.	*Rot.*	*Nom*	*Obs*	*Number of flats*	*Number of prims*	*Max prim contents*
Type A	2	$x'y'$	-	1 f	1 v	1	1	1
"	2	$x'y'$	θ	1 f	1 v	6	1	6
"	3	$x'y'z'$	-	1 f	1 v	1	1	1
"	3	$x'y'z'$	θ	1 f	1 v	7	1	7
"	3	$x'y'z'$	$\phi\theta\psi$	1 f	1 v	23	1	23
Type B	2	$x'y'$	-	1 v	1 f	1	1	1
"	2	$x'y'$	θ	1 v	1 f	2	1	2
"	3	$x'y'z'$	-	1 v	1 f	1	1	1
"	3	$x'y'z'$	θ	1 v	1 f	5	1	5
"	3	$x'y'z'$	$\phi\theta\psi$	1 v	1 f	21	1	21
Type C	3	$x'y'z'$	-	1 e	1 e	1	1	1
"	3	$x'y'z'$	θ	1 e	1 e	126	10	23
"	3	$x'y'z'$	$\phi\theta\psi$	1 e	1 e	606	10	103

Table 6.1: **Set-tree composition for constraint surfaces**

Applicability cone	*Dimen -sions*	*DOFs*		*Input*		*Output*		
		Trans.	Rot.	Nom	Obs	*Number of flats*	*Number of prims*	*Max prim contents*
Type A	2	$x'y'$	θ	1 f	1 v	8	2	4
"	3	$x'y'z'$	θ	1 f	1 v, k e	$4k$	k	4
"	3	$x'y'z'$	$\phi\theta\psi$	1 f	1 v, k e	$20k$	k	20
Type B	2	$x'y'$	θ	1 v	1 e	8	2	4
"	3	$x'y'z'$	θ	1 v, k e	1 e	$4k$	k	4
"	3	$x'y'z'$	$\phi\theta\psi$	1 v, k e	1 e	$20k$	k	20

Table 6.2: **Set-tree composition for applicability cones**

C-space obstacle	*Dimen -sions*	*DOFs*		*Input*		*Output*		
		Trans.	Rot.	Nom	Obs	*Number of flats*	*Number of prims*	*Max prim contents*
Bounded convex case	2	$x'y'$	-	n-gon	m-gon	$2mn$	$2mn$	1
"	2	$x'y'$	θ	n-gon	m-gon	$8mn$	$2mn$	6
"	3	$x'y'z'$	-	n-hedra	m-hedra	$13mn - 22n - 22m + 36$	$13mn - 22n - 22m + 36$	1
"	3	$x'y'z'$	θ	n-hedra	m-hedra	$1158mn - 2296n - 2288m + 4536$	$94mn - 184n - 184m + 360$	23
"	3	$x'y'z'$	$\phi\theta\psi$	n-hedra	m-hedra	$5542mn - 11000n - 10992m + 21816$	$94mn - 184n - 184m + 360$	103
Unbounded convex case	2	$x'y'$	-	n-gon	m-gon	$2mn$	$2mn$	1
"	2	$x'y'$	θ	n-gon	m-gon	$24mn$	$6mn$	6
"	3	$x'y'z'$	-	n-hedra	m-hedra	$13mn - 22n - 22m + 36$	$13mn - 22n - 22m + 36$	1
"	3	$x'y'z'$	θ	n-hedra	m-hedra	$1206mn - 2344n - 2336m + 4536$	$106mn - 196n - 196m + 360$	23
"	3	$x'y'z'$	$\phi\theta\psi$	n-hedra	m-hedra	$5782mn - 11240n - 11232m + 21816$	$106mn - 196n - 196m + 360$	103

Table 6.3: **Set-tree compositions for C-space obstacles for convex objects**

Finally, note that Tables 6.3 and 6.4 correspond to *convex* cases. For cases where the nomad consists of i convex products, and the obstacle has j convex products, the C-space obstacle will consist of ij products where the complexity of each product can be calculated from the above table.

Operation	*Dimen -sions*	*DOFs*		*Input*		*Output*		
		Trans.	Rot.	Nom	Obs	*Number of flats*	*Number of prims*	*Max prim contents*
CS-obs for bounded convex case	2	$x'y'$	-	rectangle	rectangle	32	32	1
"	2	$x'y'$	θ	rectangle	rectangle	128	32	6
"	3	$x'y'z'$	-	cuboid	cuboid	240	240	1
"	3	$x'y'z'$	θ	cuboid	cuboid	18720	1536	23
"	3	$x'y'z'$	$\phi\theta\psi$	cuboid	cuboid	89376	1536	103
CS-obs for unbounded convex case	2	$x'y'$	-	rectangle	rectangle	32	32	1
"	2	$x'y'$	θ	rectangle	rectangle	256	96	6
"	3	$x'y'z'$	-	cuboid	cuboid	240	240	1
"	3	$x'y'z'$	θ	cuboid	cuboid	19872	1824	23
"	3	$x'y'z'$	$\phi\theta\psi$	cuboid	cuboid	95136	1824	103

Table 6.4: **Set-tree composition for two specific cases—rectangles in two dimensions ($n = m = 4$) and cuboids in three dimensions ($n = m = 6$)**

6.10 Test results

All of the precise global C-space mapping algorithms described above were implemented in C++ using the svLis-m geometric modelling kernel. None of the algorithms have been optimised, and in particular a significant proportion of the times given are used in printing information regarding the construction process.

Table 6.6 reports the CPU time and maximum memory requirement for a number of tests. Tests were executed on a Silicon Graphics Onyx workstation, with two 150 MHz MIPS R4400 processors and 256 MB of memory, except where an asterisk (*) accompanies the result, which indicates the test was executed on a Silicon Graphics Origin server with twenty 196 MHz MIPS R10000 processors and 6 GB of memory.

The column headed 'B' indicates whether or not the mapmaker was given the hint that the objects were bounded—a 'B' entry indicates the mapmaker was given such a hint, a 'U' indicates the mapmaker had to treat the objects as though they were unbounded.

Where images of the C-space map were unobtainable, results are illustrated by snapshots from an exploration of the C-space map using `Orienteer` (Chapter 4). In each case, two snapshots are providing, showing two configurations which are very close together. Notice the indicator in the bottom left corner of each snapshot which shows green when a configuration has tested *safe* in the C-space map, and red when the configuration has tested *prohibited.* In each case, one of the two configurations is *safe* and the other *prohibited*, showing that the surface of the C-space obstacle in the C-space map is correctly located.

6.11 Summary & conclusions

This chapter has described an implemented algorithm for computing a precise representation of configuration-space obstacles for a significant subset of problems—polytope cases in two or three dimensions for which a relatively small convex decomposition can be obtained.

Figure 6.14: **An illustration of a convex 2-D 3-DOF case**

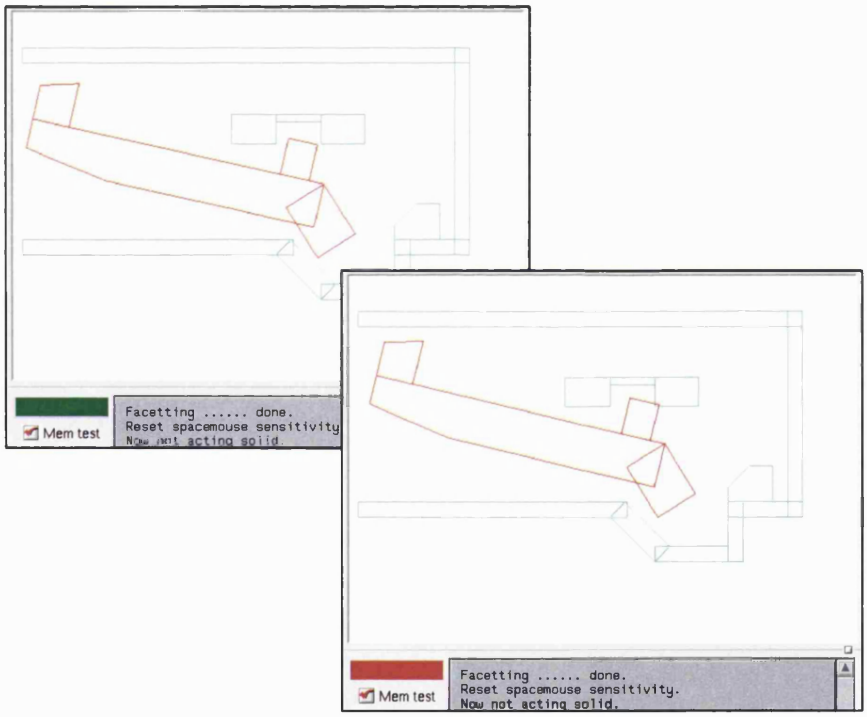


Figure 6.15: **An illustration of a non-convex 2-D 3-DOF case**

It is clear that the set-theoretic representation has many properties which suit it to analytical C-space mapping:

- The equivalence of convexity and intersection makes it straightforward to compute the C-space obstacle caused by one convex object to another.

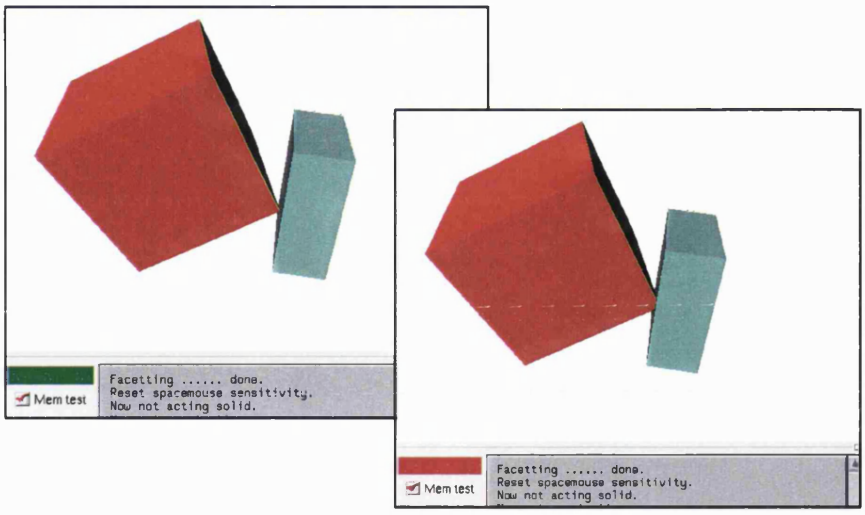


Figure 6.16: **An illustration of a convex 3-D 6-DOF case (i)**

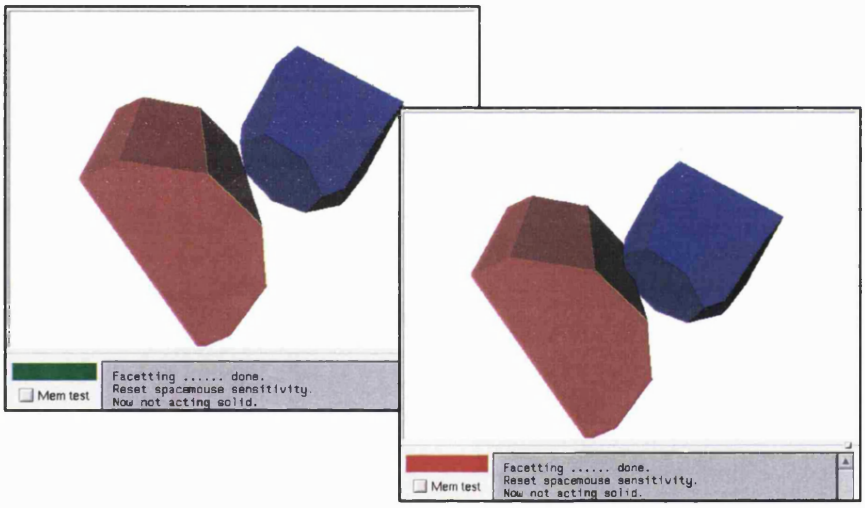


Figure 6.17: **An illustration of a convex 3-D 6-DOF case (ii)**

- The implicit property of set-theoretic modelling makes it unnecessary to calculate the intersection between contact constraints and applicability conditions—which may be complicated multidimensional surfaces. The computation of adjacency information, which is also very expensive in multidimensional space, is also avoided.

- The dimension-independent notation enables multidimensional C-space obstacles to be computed using straightforward extensions to the algorithms designed for simpler cases.

- An implicit representation can be obtained for each contact constraint

which occurs in a polytope case by substituting expanded rotation matrices into the linear halfspace basis.

- The disjunctive form offers a simple solution to convex decomposition which is practical for some interesting cases (although, in the worst case the number of products is exponential in the number of leaves in the original).

Indeed, at the outset I pointed out that the ease of conversion between the set-theoretic representation and B-rep (for polytopes) meant that our set-theoretic modeller could use the B-rep algorithms. In contrast, the success of the algorithms described in this chapter—which have been implemented to obtain precise global C-space maps of a higher dimensionality than those demonstrated in the literature—suggests that B-rep modellers might consider converting to a set-theoretic representation to compute such maps.

Case name	*Work-space*	*Obstacle composition (products×flats)*	*Nomad composition (product×flats)*	*Illustration*
Rectangles	2-D	1×4	1×4	Figure 6.14
2-D Housing & catch (simplified)	2-D	$1 \times 5, 10 \times 4$	$2 \times 5, 2 \times 4$	Figure 6.15
2-D Housing & catch (polygonised)	2-D	$1 \times 7, 5 \times 4, 5 \times 3, 2 \times 2, 2 \times 1$	$2 \times 14, 2 \times 5$	-
Cuboids	3-D	1×6	1×6	Figure 6.16
3-D Plunger head & shaft head	3-D	1×9	1×13	Figure 6.17
3-D Plunger & shaft	3-D	$1 \times 13, 2 \times 6$	5×6	Figure 6.12
3-D Housing & catch (simplified)	3-D	$1 \times 7, 10 \times 6$	$2 \times 7, 2 \times 6$	-
3-D Housing & catch (polygonised)	3-D	$1 \times 9, 5 \times 6, 5 \times 5, 2 \times 4, 2 \times 3$	$2 \times 16, 2 \times 7$	-

Table 6.5: **Test cases for the analytical C-space mapping algorithms**

Case	*DOFs*	*B?*	*CPU time*	Max. *memory usage*	*Illustration*
Cuboids	$x'y'z'$	B	0.28 s.	0.6 MB	-
Cuboids	$x'y'z'$	U	0.42 s.	1.3 MB	-
3-D Plunger & shaft	$x'y'z'$	B	9.69 s.	1.3 MB	-
3-D Plunger & shaft	$x'y'z'$	U	15.42 s.	5.6 MB	Figure 6.12
Rectangles	$x'y'\theta$	B	0.42 s.	3.7 MB	-
Rectangles	$x'y'\theta$	U	0.46 s.	5.5 MB	Figure 6.14
2-D Housing & catch (simplified)	$x'y'\theta$	U	10.76 s.	8.2 MB	Figure 6.15
2-D Housing & catch (polygonised)	$x'y'\theta$	U	25.45 s.	7.6 MB	-
Cuboids	$x'y'z'\theta$	U	6.14 s.	3.6 MB	-
Cuboids	$x'y'z'\phi\theta\psi$	U	12.90 s.	16.3 MB	Figure 6.16
3-D Plunger head & shaft head	$x'y'z'\phi\theta\psi$	U	70.90 s.*	76.8 MB *	Figure 6.17
3-D Plunger & shaft	$x'y'z'\phi\theta\psi$	U	98.37s. *	301.2 MB *	-
3-D Housing & catch (simplified)	$x'y'z'\phi\theta\psi$	U	131.81 s. *	354.1 MB *	-
3-D Housing & catch (polygonised)	$x'y'z'\phi\theta\psi$	U	416.99 s.	881.6 MB *	-

Table 6.6: **Test results for the precise C-space mapping algorithms for a single nomad**

Chapter 7

C-space mapping for multiple nomads

7.1 Introduction

A survey of C-space applications beyond single robot problems (Wise and Bowyer [106]) revealed that very few papers have focussed on C-space mapping for multiple independent objects, which is inevitable given the difficulties in representing highly dimensional C-space obstacles. However, as described in Chapter 5, the omni-model approach works (in principle at least) independently of the number of objects.

Moreover, this Chapter describes how the analytical methods described in Chapter 6 have been used to obtain global C-space maps for problems involving a greater number of nomads than any found in the existing literature.

These results are significant to the long-term goals of our research (discussed in Section 9.8) since a rigid-body mechanism is equivalent to a multiple-nomad problem.

7.2 Calculating the C-space obstacles caused by static obstacles

The configuration space for multiple nomads is the product of all the degrees of freedom of the nomads. The C-space obstacle caused by a static obstacle, OBS to an individual nomad, $\mathsf{NOM_i}$, can be calculated using the appropriate algorithm from Chapter 6, as if $\mathsf{NOM_i}$ was the only nomad. $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{NOM_i}}$ is then embedded into the higher dimensional C-space. Since none of the flats which form $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{NOM_i}}$ have coefficients in the dimensions relating to the degrees of freedom of the other nomads, $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{NOM_i}}$ will extrude orthogonally into those additional dimensions. When a configuration point or box is tested against $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{NOM_i}}$, the values in the additional dimensions have no effect on the potential value returned.

The C-space obstacle caused to each nomad by the set of static obstacle is calculated individually and all the results are unioned.

7.3 Calculating the C-space obstacles caused by other nomads

Consider a problem with m nomads, $\mathsf{NOM_0}, \mathsf{NOM_1}, \ldots \mathsf{NOM_{m-2}}, \mathsf{NOM_{m-1}}$. The C-space obstacles that the nomads cause to each other are computed as follows:

1. For each nomad, $\mathsf{NOM_i}$, compute the C-space obstacles caused by interactions with each $\mathsf{NOM_j}$ where $j = i+1, i+2, \ldots, m-2, m-1$.

   Each $\mathcal{CSOBS}^{\mathsf{NOM_j}}_{\mathsf{NOM_i}}$ is computed as follows:

   (a) The appropriate algorithm from Chapter 6 is used to compute the C-space obstacle caused by $\mathsf{NOM_j}$ when it is a static obstacle in its initial configuration. This C-space obstacle is formulated in terms of the degrees of freedom of $\mathsf{NOM_j}$.

   (b) The resulting $\mathcal{CSOBS}^{\mathsf{NOM_j}}_{\mathsf{NOM_i}}$ is made to move as $\mathsf{NOM_j}$ moves, by performing the same rotational and translational sweeps which are used to make an omnimodel. The dimensions used to represent the extra

degrees of freedom are the ones already used to represent the degrees of freedom of $\mathsf{NOM_j}$. Thus, the new swept version of $\mathcal{CSOBS}^{\mathsf{NOM_j}}_{\mathsf{NOM_i}}$ is formulated in both groups of degrees of freedom.

(c) If $\mathsf{NOM_j}$ has any rotational degrees of freedom, $\mathcal{CSOBS}^{\mathsf{NOM_j}}_{\mathsf{NOM_i}}$ will currently be incorrect since as $\mathsf{NOM_j}$ rotates, the orientation of $\mathsf{NOM_i}$ relative to $\mathsf{NOM_j}$ changes—thus the C-space obstacle changes shape as well as rotating. This is taken into account by replacing all the absolute orientation values with relative orientations—for example, within the set tree, each instance of $\theta_{\mathsf{NOM_j}}$ at a leaf is replaced by $(\theta_{\mathsf{NOM_i}} - \theta_{\mathsf{NOM_j}})$.

The resulting $\mathcal{CSOBS}^{\mathsf{NOM_j}}_{\mathsf{NOM_i}}$ models the C-space obstacle the two nomads cause each other; it is formulated in terms of the degrees of freedom of $\mathsf{NOM_i}$ and $\mathsf{NOM_j}$, and is embedded into the higher dimensional C-space of the complete problem, where it projects orthogonally in the additional dimensions.

2. All the $\mathcal{CSOBS}^{\mathsf{NOM_j}}_{\mathsf{NOM_i}}$ computed in this way are unioned together and placed into a box which bounds all the degrees of freedom of the nomads. The result is a complete and accurate C-space map of the multiple-nomad problem.

7.4 Handling unbounded objects, non-convex objects and three-dimensional objects

The above algorithm for computing a C-space map for a multiple nomads extends in exactly the same way as the single-nomad algorithm described in Chapter 6—it therefore has the same strengths and weaknesses:

Handling unbounded objects Every $\mathcal{CSOBS}$ computed in the algorithm is based on a $\mathcal{CSOBS}$ computed using the single-nomad algorithm. Thus, unbounded objects can be handled by introducing applicability conditions as sets. These ensure correct results and introduce the possibility of a divide-and-conquer methodology. However, as before, these typically more than double the number of flats in the result.

Handling non-convex objects Non-convex objects are broken into convex pieces using the Disjunctive Form and each interaction is modelled in a pair-wise fashion. As before this is tractable for relatively simple cases, but in the worst case the Disjunctive Form generates an exponential number of products.

Handling three-dimensional objects For translations and rotations about the z axis, the multiple-nomad algorithm extends to three-dimensional environments simply by making calls to the three-dimensional version of the single-nomad algorithm. As before this is expensive because there are typically more Type A and Type B interactions than the two-dimensional equivalent, and type C interactions, which are particularly complicated, are introduced.

Note that Step 1c of the algorithm incorporates the relative orientation of one nomad to another into a set tree. If the nomads have full rotational freedom in three dimensions, the relative orientation corresponds to the product of two rotation matrices, so the expression becomes considerably more complicated. This is discussed in future work.

7.5 Complexity analysis

A case involving a convex obstacle set and m convex nomads requires $\begin{pmatrix} m+1 \\ 2 \end{pmatrix}$ C-space obstacles to be computed, where:

$$\begin{pmatrix} m+1 \\ 2 \end{pmatrix} = \frac{(m+1)!}{2.(m-1)!} = \frac{1}{2}m(m+1)$$

The composition of each of these C-space obstacles can be calculated from Section 6.9, and independent of the number of nomads. Thus, the multiple nomad algorithm is $O(m^2)$ in the number of nomads.

7.6 Experimental results

7.6.1 Test platform

The multiple-nomad algorithm described in this chapter has been implemented (using the svLis-m geometric modelling kernel) for cases involving 2-D nomads with three degrees of freedom. All tests reported were executed on a Silicon Graphics Onyx workstation, with two 150 MHz MIPS R4400 processors and 256 MB of memory.

7.6.2 Test results

Confirmation of correct behaviour

Figure 7.1 shows snapshots from a session using `Orienteer` (Chapter 4) to explore the C-spaces of a case with six independent 2-D nomads with three degrees of freedom each. Note that in (a) the indicator in the bottom left indicates that the configuration is *safe* whilst in (b), where one of the nomads has been moved to overlap one of the others, the indicator correctly indicates *prohibited.*

Thus, precise, global C-space maps have been constructed of up to eighteen dimensions, and have been demonstrated to be correct.

The effect of the number of nomads

Figures 7.2 and 7.3 plot time and memory requirement against the number of nomads for two cases—one involving convex nomads, the other non-convex. These results show that even in an unoptimised form, the algorithm can compute a precise global eighteen-dimensional C-space map in approximately a minute, requiring approximately 30 MB of memory.

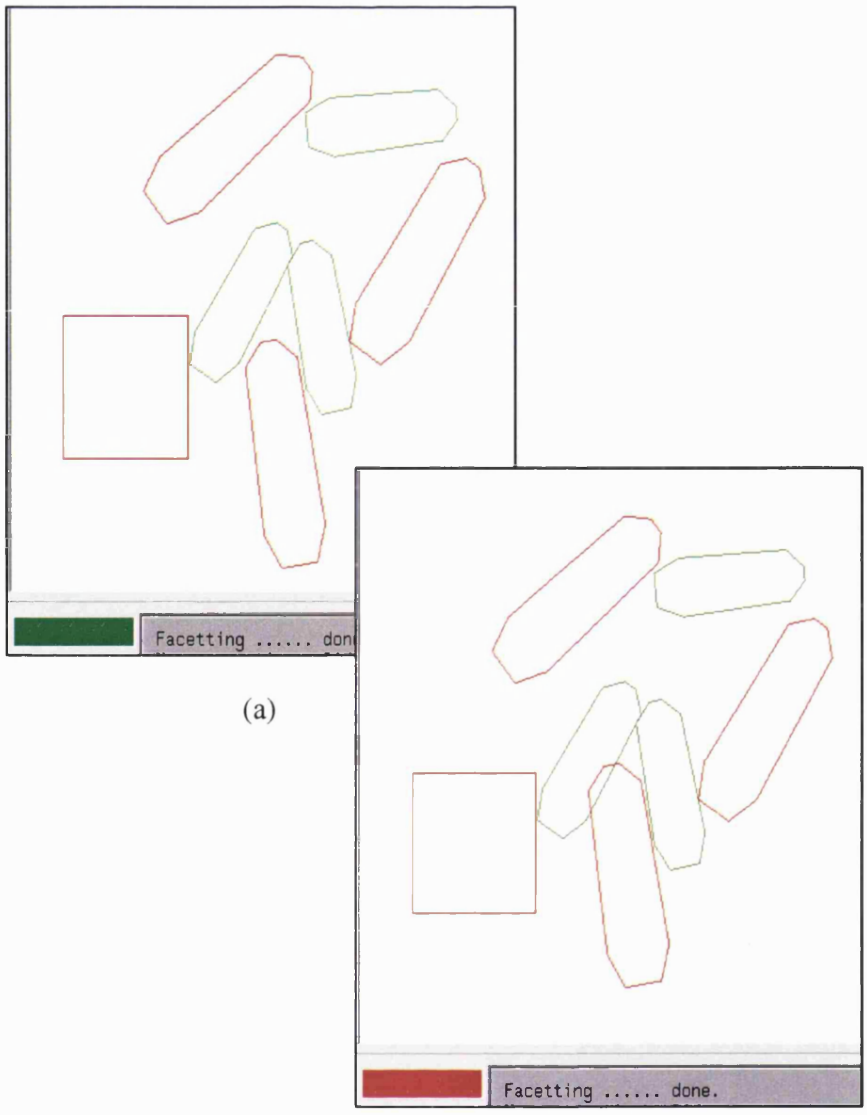


Figure 7.1: **Orienteer explores the C-space of a case with six 2-D nomad with three DOFs each**

7.7 Summary & conclusions

The precise C-space mapping algorithm for polytopes described in Chapter 6 extends to multiple-nomads in a straightforward manner: the C-space obstacle which nomad B causes to nomad A is computed as if B is a static obstacle, then that C-space obstacle is swept in the multidimensional space such that it effectively moves as B moves. After applying this sweep, which uses the same transformations as those used to construct an omnimodel, allowance is made for

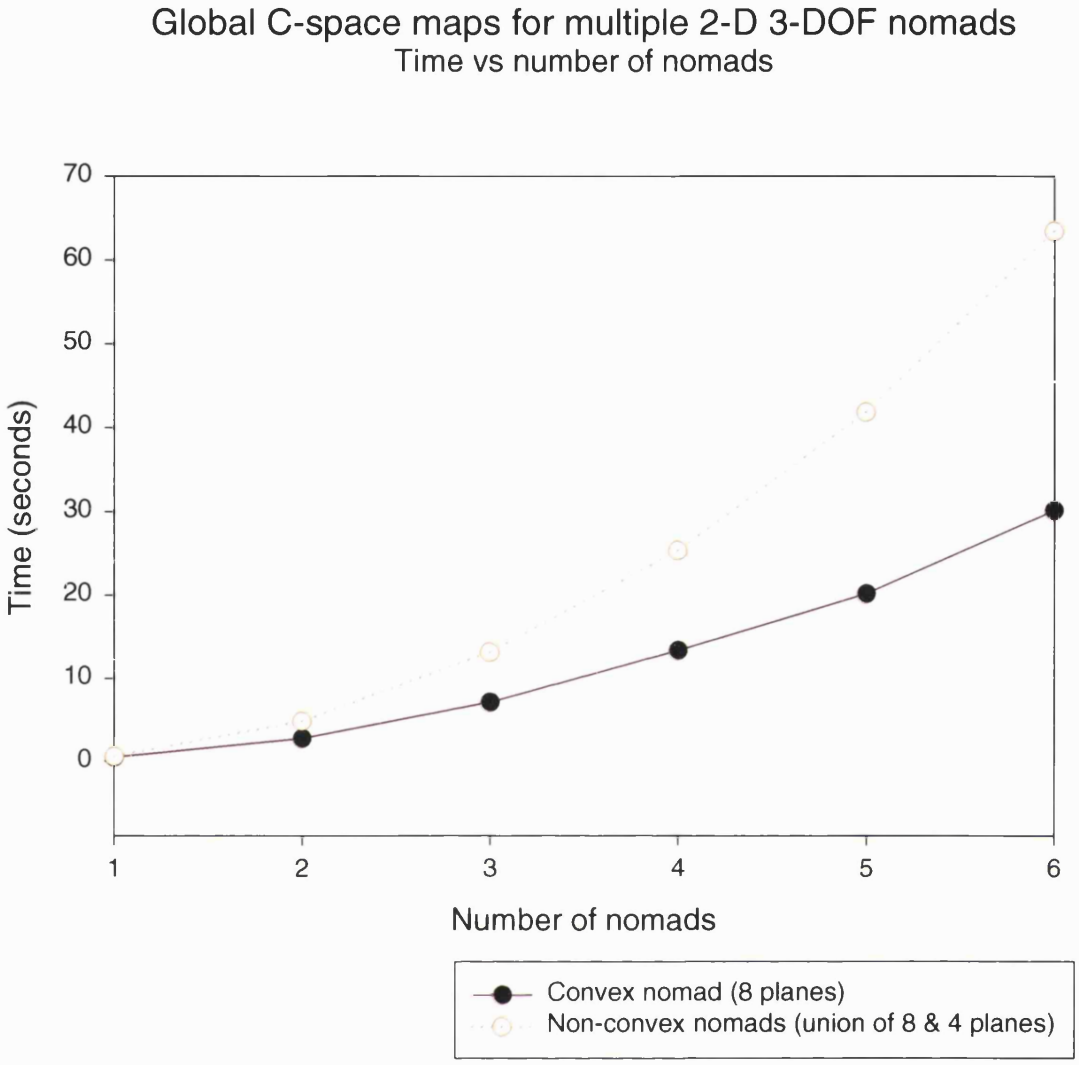


Figure 7.2: **Time taken plotted against the number of nomads for convex and non-convex cases**

the relative orientation between two nomads, which depends upon both sets of rotational degrees of freedom.

This approach has been tested by implementing the algorithm for two-dimensional cases with three degrees of freedom. The algorithm, which was straightforward to implement using svLis-m, has successfully computed precise global C-space maps with up to eighteen dimensions (higher than any global map found in the literature) and only takes approximately a minute to do so.

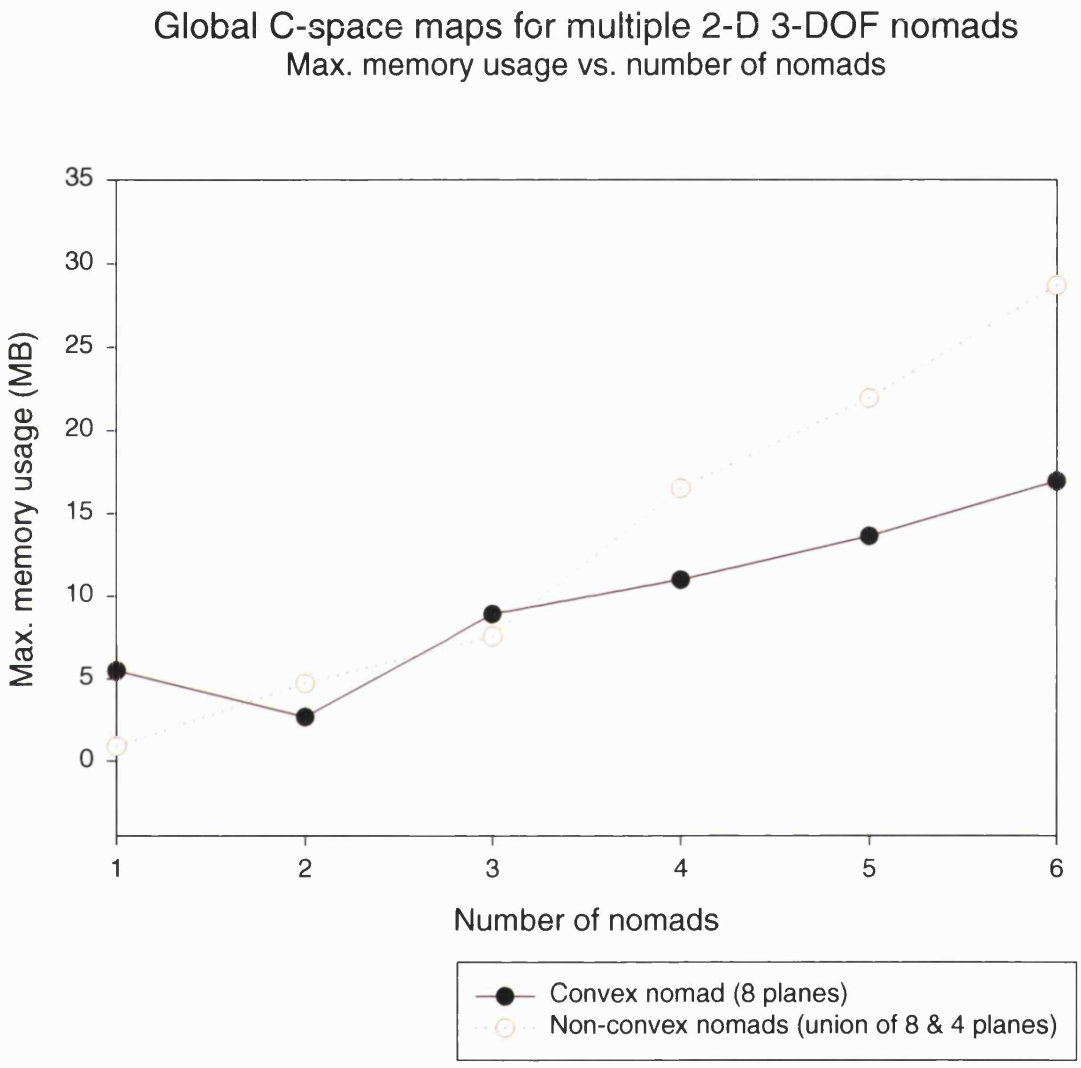


Figure 7.3: **Max. memory requirement plotted against the number of nomads for convex and non-convex cases**

Chapter 8

C-space mapping for a manipulator arm

8.1 Introduction

As mentioned earlier, the links of a robotic manipulator can be treated as a series of connected rigid objects, each of which has one or more degrees of freedom *relative to the link below it* (typically, each link will have one prismatic or revolute degree of freedom). The degrees of freedom of the whole manipulator (which define the configuration space) are the sum of the degrees of freedom of all links.

As the literature survey shows, more papers focus on mapping the C-space for a single robotic manipulator in a static environment than any other C-space mapping domain. Most of the algorithms surveyed divided the C-space into axially-aligned cells (regular grids or 2^n-trees), whilst techniques which produce only a partial map are playing an increasing role.

This Chapter describes three approaches which can be used to map the configuration-space of a robotic manipulator using multidimensional set-theoretic modelling:

1. The C-space can be mapped for the links as if they were multiple independent nomads (Chapter 7), and then each link can be constrained relative

to the previous link by introducing constraint sets into the C-space (Eisenthal [27]).

2. By taking a change of perspective, the mapmaking techniques from Chapters 6 and 7 can be adapted such that the C-space map for the manipulator can be computed directly.

3. The omnimodel method of Chapter 5 can be extended by taking the same 'change of perspective' as the second approach.

As shown later (Chapter 8.5) an implementation of the second approach has produced, for polygonal environments, C-space maps which are complete, precise and of higher dimensionality than any of those found in the existing literature. Results have also been obtained for polyhedral environments where all joints are rotational about the **z** axis.

8.2 Combining C-space mapping with constraint modelling

Consider the two-dimensional approximation of a simple manipulator illustrated in Figure 8.1 (a). Each of the two polygonal links, LINK_0 and LINK_1, have one rotational degree of freedom each, θ_0 and θ_1 respectively,,and the manipulator operates in a polygonal environment. Figure 8.1 (b) shows the manipulator in its initial (reference) configuration whilst (c) and (d) show each link in its own model space (that is, its local coordinate system). Notice that the reference point for each link has been chosen as the point where the link will be attached to the previous link. Notice also that LINK_0 has an upper joint location, u_0, associated with it—that is where $LINK_1$ will be attached.

Figure 8.1 (e) shows the static obstacles, including the base. The point where LINK_0 is attached to the base is labelled b.

The C-space obstacles caused by the static obstacles to LINK_0 can be computed easily:

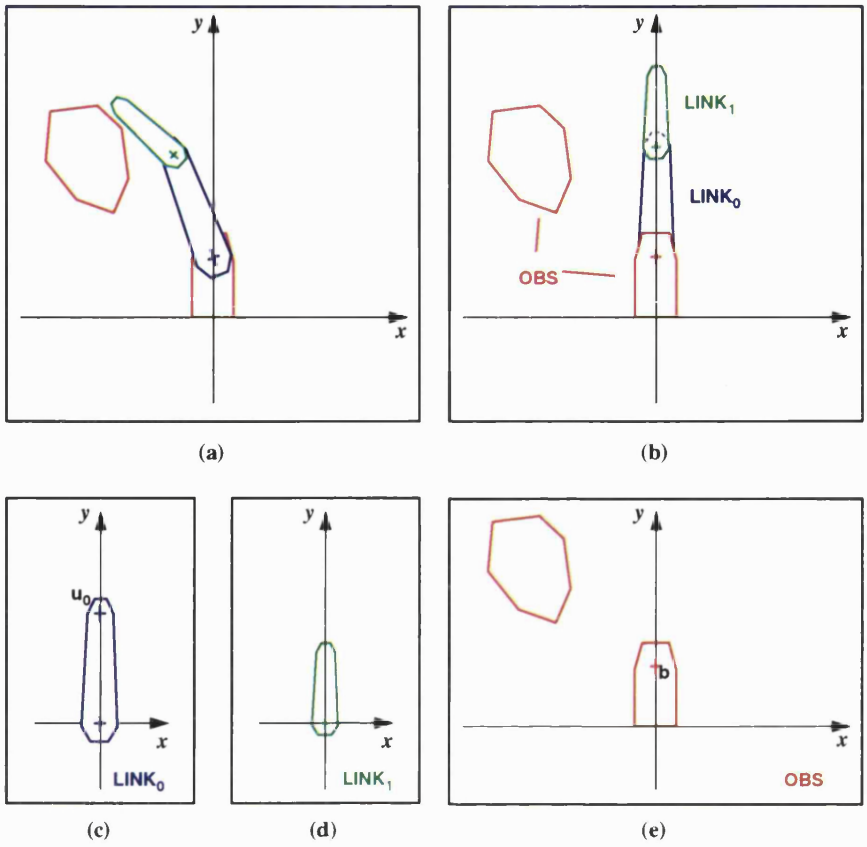


Figure 8.1: **A 2-D approximation of a 2-link manipulator**

1. Using one of the algorithms described in Chapter 6, $\mathsf{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_0}$ is computed as if LINK_0 was an independent nomad with a full three degrees of freedom.

2. The two translational degrees of freedom, $\mathbf{x}'_0$ and $\mathbf{y}'_0$, (which LINK_0 does not have since it is attached to the base at point b) are removed. This is achieved by slicing the svLis-m model representing the C-space map at the point $(\mathbf{x}'_0 : x^b, \mathbf{y}'_0 : y^b)$ (see Section 3.4.9).

3. The result is a model of the one-dimensional C-space. This consists of an interval and a set-tree whose leaf flats are all one-dimensional flats (i. e. they all have a normal of $(\theta_0 : 1)$ or $(\theta_0 : -1)$).

Clearly, although LINK_1 has the same relative degree of freedom as LINK_0, computing the C-space obstacles caused to it by the static obstacles is less straightforward. The translational degrees of freedom cannot be sliced away in the same fashion since, in order to reach configurations such as that shown in Figure 8.1 (a),

Link 1 must translate as well as rotate from its initial configuration (Figure 8.1 (d)). However, the set of safe configurations for the link (and consequently the manipulator) can be obtained by treating the manipulator as a hybrid problem involving C-space and geometric constraints.

As detailed by Eisenthal [27], svLis-m can be used to model multidimensional geometric constraints by treating them as sets which are intersected in the configuration space. For example, consider the constraint that the rotational joint between LINK_0 and LINK_1 imposes on LINK_1: it fixes the reference point of LINK_1 to always coincide with u_0. So, for all valid configurations:

$$\begin{aligned} \mathbf{x}'_1 &= x^{u_0} \\ \mathbf{y}'_1 &= y^{u_0} \end{aligned} \tag{8.1}$$

The position of u_0 can be formulated in terms of the one degree of freedom of LINK_0:

$$\begin{aligned} \mathbf{x}^{u_0} &= \overline{u_0} \cos \theta_0 \\ \mathbf{y}^{u_0} &= \overline{u_0} \sin \theta_0 \end{aligned}$$

where $\overline{u_0}$ is the modulus of u_0 when LINK_0 is in its initial configuration:

$$\overline{u_0} = \sqrt{(\mathbf{x}_0^{u_0})^2 + (\mathbf{y}_0^{u_0})^2} \tag{8.2}$$

Combining 8.1 and 8.2 tells us that at valid configurations:

$$\begin{aligned} \mathbf{x}'_1 &= \overline{u_0} \cos \theta_0 \\ \Rightarrow \mathbf{x}'_1 - \overline{u_0} \cos \theta_0 &= 0 \end{aligned}$$

This constraint can be represented in svLis-m by the tree shown in Figure 8.2 where the 'abs' operator (Section 3.2.6) forces the potential value of the set to be positive everywhere except for the surface—forming a sheet of zero thickness.

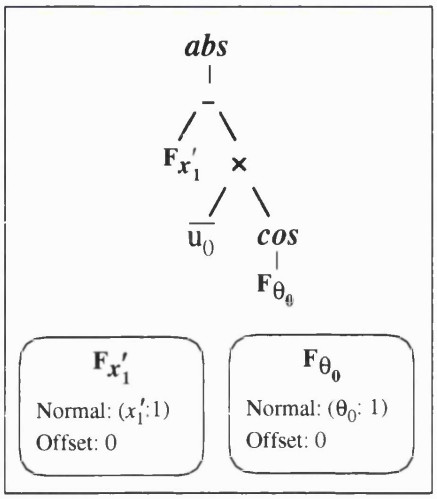


Figure 8.2: **A set-tree representing a geometric constraint**

When this sheet is intersected with the equivalent constraint on $\mathbf{y}_1'$, the result is a one-dimensional helical wire embedded in a three-dimensional C-space, as illustrated in Figure 8.3.

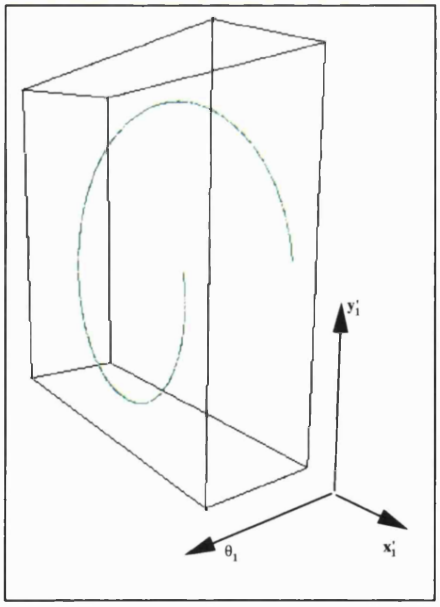


Figure 8.3: **A 'wire' formed by intersecting thin sheets**

So, by applying the multiple-nomad algorithm to the manipulator links we have

obtained a four-dimensional C-space map, and by modelling explicit constraints we have obtained a one-dimensional wire (embedded in a three-dimensional C-space) which represents valid configurations. A representation of the $\mathcal{FREESPACE}$ of the manipulator can now be obtained by placing the constraint 'wire' into the four-dimensional C-space (where it projects orthogonally in the additional dimension to form a sheet) and intersecting it with the complement of the C-space obstacles.

This representation of $\mathcal{FREESPACE}$ is complete and precise, and can be explored by sliding the configuration around the sheet using a variety of methods (see Eisenthal [27]).

8.3 Direct computation of $\mathcal{FREESPACE}$

8.3.1 Mapmaking from a different perspective

Although complete, the representation of $\mathcal{FREESPACE}$ obtained by the previous method is inefficient because of the artificially high dimensionality. The problem is that the additional dimensions cannot be removed since a model can only be sliced to a point (by fixing chosen dimensions at specific values), not to a helical wire.

To explain how this problem is sidestepped by taking a 'change of perspective' let us reconsider the problem of computing the C-space obstacles which the obstacles cause to LINK_1 for the case illustrated in Figure 8.1.

We want to remove the translation dimensions $\mathbf{x}'_1$ and $\mathbf{y}'_1$ from our final C-space map by slicing it at a point ($(\mathbf{x}'_1 : 0, \mathbf{y}'_1 : 0)$, say) but doing so has the effect of treating LINK_1 as though it does not translate from its initial configuration. We must therefore construct the C-space obstacles as if LINK_1 stays still and, where necessary, move everything else relative to it. The procedure is:

1. Translate the obstacles by a vector $-(b + u_0)$ to place them into LINK_1's

model-space[1]. This is necessary since when LINK_1 is in the manipulator's initial position (Figure 8.1 (d)) it is translated by $(b + u_0)$.

This gives the model illustrated in Figure 8.4.

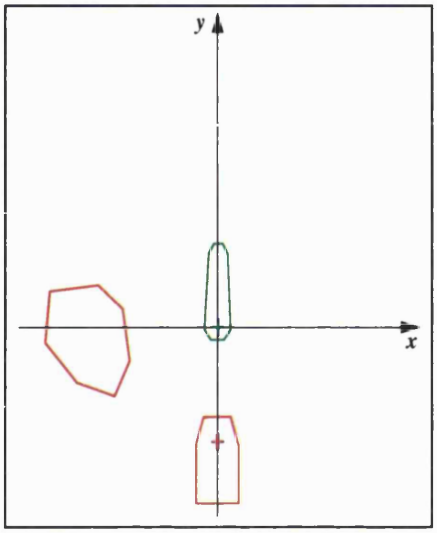


Figure 8.4: **The obstacles are placed into the model space of** LINK_1

2. Compute $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ as if LINK_1 is an independent nomad using the usual algorithm (Chapter 6). This C-space obstacle set will contain $\mathbf{x}'_1$ and $\mathbf{y}'_1$ which will be removed later but are left in place for now.

3. Incorporate the effect of LINK_0's rotation (θ_0). As θ_0 increases, LINK_1 rotates counter-clockwise about a joint which is positioned at $(-u_0)$ relative to the reference point of LINK_1. To fulfil this movement, LINK_1 has to translate—which we must avoid. Instead, we treat LINK_1 as static and make the obstacle move relative to it. Specifically, as θ_0 increases we make $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ rotate *clockwise*, about point $(-u_0)$. This rotational sweep is achieved in the usual way except that:

   (a) Since the centre of rotation is not the origin, we must:

      i. Temporarily translate the model such that the centre of rotation coincides with the origin.

      ii. Perform the rotational sweep.

      iii. Translate the model back again.

[1] Where LINK_1's *model-space* refers to a model of the workspace in which the reference point of LINK_1 is at the origin, and the rest of the objects have the correct relative position to LINK_1. It can be considered as the workspace modelled from that particular link's perspective.

(b) The direction of rotation is reversed (to clockwise) by replacing all instances of θ_0 in the set tree with $(-\theta_0)$.

4. In the set-tree of $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$, replace all instances of θ_1 (introduced at Step 2) with $(\theta_1 + \theta_0)$. This is necessary since as θ_0 changes, so does the orientation of LINK_1 relative to the static obstacles.

   $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ is now a correct representation of the C-space obstacles that the obstacles caused to LINK_1 formulated in $\mathbf{x}'_1$, $\mathbf{y}'_1$, θ_0, and θ_1.

5. Slice $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ at the point $(\mathbf{x}'_1 : 0, \mathbf{y}'_1 : 0)$ to leave a two-dimensional C-space obstacle. The result is still a correct representation of the C-space obstacles, since we took care to ensure that LINK_1 was treated as if it did not translate.

6. Union $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ with $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_0}$ (computed as described in Section 8.2), and place it in a box bounding both degrees of freedom (Figure 8.5). This is a C-space map for the manipulator (ignoring link-link interactions). To illustrate how the C-space map corresponds to the robot in the workspace, Figure 8.5 shows the robot in three configurations corresponding to key points in the map. In configuration (a), the orientation of LINK_1 places the maximum limit on the anti-clockwise rotation of LINK_0—rotating LINK_1 in either direction gradually allows LINK_0 to rotate anti-clockwise further. The orientation of LINK_1 also effects the limit on the clockwise motion of LINK_0. However, the placement of the obstacle is such that LINK_1 can only interfere with the obstacle for a small range of rotation—between configurations (b) and (c).

8.3.2 Modelling additional links

Consider the introduction of an additional link, LINK_2, as illustrated in Figure 8.6.

$\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_0}$ and $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ can be computed exactly as before. Computing $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_2}$ requires only minor adaptations:

1. In order to place the obstacles into the model space of LINK_2, they must be translated by $-(b + u_0 + u_1)$ (where u_k is the upper joint location of

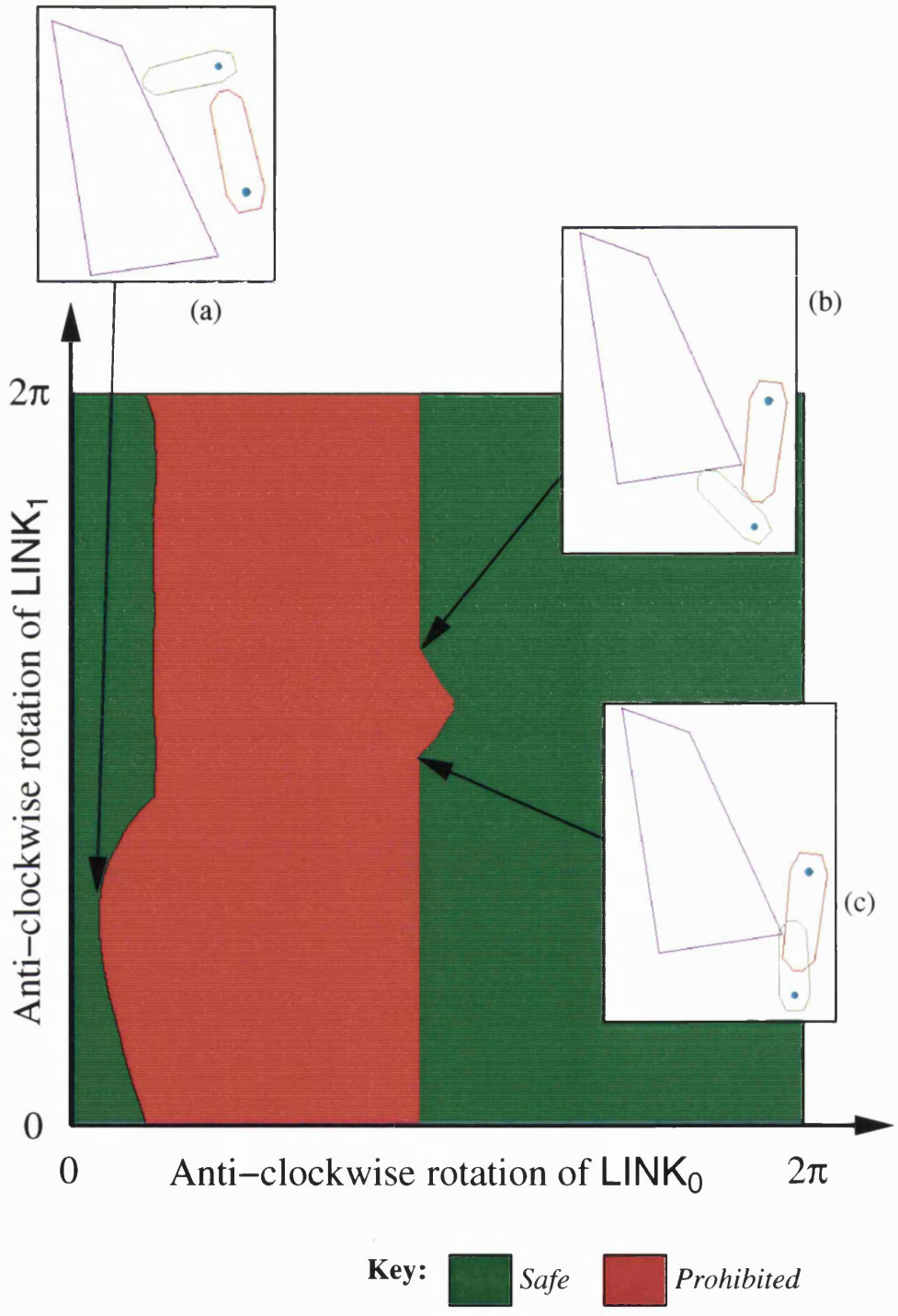


Figure 8.5: **The C-space map for a manipulator similar to that of Figure 8.1, ignoring link-link interactions**

LINK_k, that is, the point in the model space of LINK_k where LINK_{k+1} will be attached).

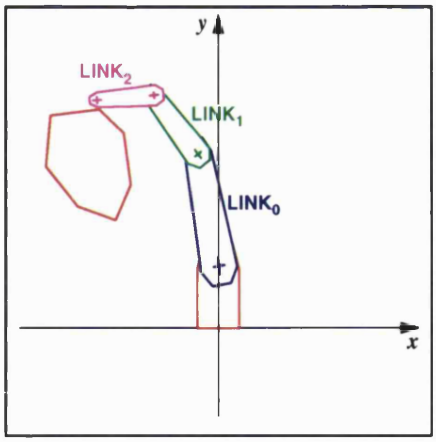


Figure 8.6: **A 2-D approximation of a 3-link manipulator**

2. As before, compute $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_2}$ as if LINK_2 is an independent nomad.

3. Sweep $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_2}$ *clockwise* as θ_0 increases. The centre of rotation of this sweep is now the point $-(u_0 + u_1)$ since that is the offset from joint 0 to the reference point of LINK_2 in its initial configuration.

4. As an additional step, sweep $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_2}$ *clockwise* as θ_1 increases, with the centre of rotation at point $-(u_1)$.

5. In the set-tree of $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$, replace all instances of θ_2 (introduced at Step 2) with $(\theta_2 + \theta_1 + \theta_0)$.

6. Slice $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_2}$ at the point $(\mathbf{x}'_2 : 0, \mathbf{y}'_2 : 0)$ to leave a three-dimensional C-space obstacle.

The C-space map for the manipulator (ignoring link-link interactions) is obtained by unioning $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_2}$ with $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_1}$ and $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_0}$, and placing the result in a box bounding the three degrees of freedom.

Further links are added in the same manner—the C-space obstacle is computed, swept around the joint of each lower link, altered to allow for the cumulative affect of the joints on the link's orientation, and sliced.

8.3.3 Incorporating link-link interactions

Few of the surveyed C-space mapping papers incorporate link-link interactions. However, avoiding such collisions can be a significant problem, especially for manipulators with a high number of links. To obtain a C-space map which includes link-link interactions, the above algorithm is extended as follows:

1. Model link-link interactions in a pairwise fashion, that is, for each link, LINK_i, compute the C-space obstacle caused to it each lower link LINK_j (for all $j < i$). Each $\mathcal{CSOBS}^{\mathsf{LINK}_j}_{\mathsf{LINK}_i}$ is computed by:

   (a) treating LINK_j (in its initial configuration) as a static obstacle.

   (b) During this computation, the manipulator is modelled as if it only has the links LINK_{j+1}, LINK_{j+2}, ...,LINK_{i-1},LINK_i—any links below LINK_j are ignored because they do not affect the relative position of LINK_i to LINK_j .

2. All the $\mathcal{CSOBS}^{\mathsf{LINK}_j}_{\mathsf{LINK}_i}$ are unioned together, along with all the $\mathcal{CSOBS}^{\mathsf{OBS}}_{\mathsf{LINK}_i}$

Figure 8.7 shows a C-space computed in this way—the constraint that the shape of LINK_0 places on LINK_1 results in a horizonal stripe of constaint height across the C-space map. Note that this constraint means that LINK_1 can only touch the obstacle (and thus contribute to the C-space obstacle) for a very small amount when LINK_0 is at its most clockwise configuration.

8.3.4 Handling unbounded objects, non-convex objects and three-dimensional objects

Clearly, like the multiple-nomad algorithm, the manipulator algorithm is build around the single-nomad algorithm described in Chapter 6. Thus, cases involving unbounded objects, non-convex objects and three-dimensional objects can be solved, but with associated computational expense.

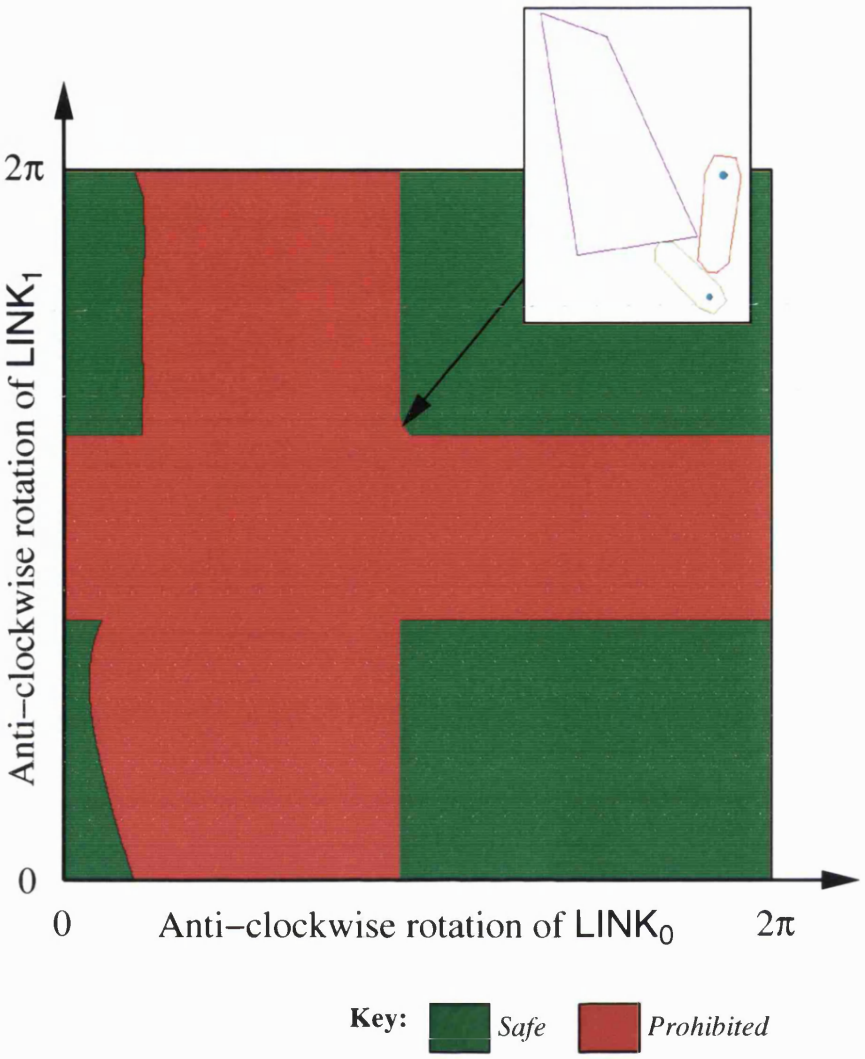


Figure 8.7: **The C-space map for a manipulator similar to that of Figure 8.1, including link-link interactions**

8.3.5 Handling other joint types

The manipulator algorithm has been implemented for two- and three-dimensional manipulators with revolute joints which rotate about the **z** axis. In principle, the algorithm extends to handle other joint types (revolute joints which rotate about the **x** or **y** axes, prismatic joints, and spherical joints) in a straightforward manner. However, these extensions are left for future work so they are discussed in Chapter 9.

It is worth noting that implementing the algorithm for revolute joints which rotate

about the **x** or **y** axis would be particularly easy, were it not for a limitation of the particular flavour of Euler angles we use to represent rotations: remember (Section 5.3.2) that no single parameter in Roll-Pitch-Roll Euler angles can be used to represent a rotation about either of these axes.

8.4 Complexity analysis

In order to analyse the complexity of the above algorithms in terms of the number of links of the manipulator, consider the case where all links are identical.

8.4.1 Ignoring link-link interactions

As described above, the first link of the manipulator is modelled by computing the C-space obstacle which one 3-DOF object causes to another, and then slicing the C-space obstacle such that the translational degrees of freedom are removed. Let c refer to the time/memory requirement for the combined construction and slicing operations[2].

When the second link is introduced, the C-space obstacle computed for the first link is still required, and that is added to by computing a second C-space obstacle in the same way. However, the second obstacle must have a rotational sweep performed on it, to account for fact that the orientation of the first link changes the position of the second. Let r refer to the time/memory requirement for the rotational sweep operation.

When a third link is introduced, the increase in time/memory is the same except two rotational sweeps must be performed to account for the two lower links.

Thus, for N links, the time/memory requirement for the algorithm is given by:

[2]Note that since all the work done by the algorithm is in constructing a tree representation within memory, time and memory complexity are the same.

$$\sum_{i=1}^{N} c + (i-1)r = \frac{N}{2}(2c + (N-1)r)$$

Thus, the time complexity of the algorithm is $O(N^2)$.

8.4.2 Modelling link-link interactions

When link-link interactions are modelled, each time a link is added, the algorithm computes the C-space obstacle which each lower link causes to it. Thus, when the ith link is added, $i-1$ C-space obstacles must be computed.

LINK_i has exactly one degree of freedom relative to LINK_{i-1} (i. e. the joint between them)), whilst each successively lower link has one more relative degree of freedom. Thus, computing the C-space obstacle each link causes to LINK_i is like computing the C-space obstacles the static obstacles cause to manipulator—each link has a corresponding C-space obstacle which undergoes one more rotational sweep than the previous.

Since each *additional* link requires as much work as the whole manipulator did previously, the complexity of the algorithm which models link-link interactions can be given as:

$$\begin{aligned} & \sum_{i=1}^{N}\sum_{j=1}^{i} c + (j-1)r \\ = & \sum_{i=1}^{N} \tfrac{i}{2}(2c + (i-1)r) \\ = & \sum_{i=1}^{N} ic + \tfrac{1}{2}\sum_{i=1}^{N} i^2 r - \tfrac{1}{2}\sum_{i=1}^{N} ir \end{aligned}$$

The most significant term of this is $\frac{1}{2}\sum_{i=1}^{N} i^2 r$, which is $O(n^3)$ (Toft [96, p. 5]).

8.5 Experimental results

8.5.1 Test platform

The manipulator algorithm described in this chapter has been implemented (using the svLis-m geometric modelling kernel) for cases involving 2-D and 3-D manipulators with revolute joints which rotate in the xy-plane. All tests reported were executed on a Silicon Graphics Onyx workstation, with two 150 MHz MIPS R4400 processors and 256 MB of memory.

Confirmation of correct behaviour

Figure 8.8 shows snapshots from a session using `Orienteer` (Chapter 4) to explore a C-spaces for a 2-D manipulator with six convex links connected by revolute joints. A precise, global six-dimensional C-space maps was constructed which incorporated link-link interactions, and was demonstrated to be correct by displaying the correct behaviour of the manipulator. Figure 8.9 illustrates that the algorithm has been tested on manipulators with non-convex links.

In both cases, note that the indicator in the bottom left corner shows that the surface of the C-space obstacle has been correctly mapped since configurations which cause a small amount of overlap are correctly differentiated from those which are just safe.

Figure 8.10 plots the time taken against the number of nomads for the convex and non-convex cases shown, both with and without incorporating link-link interactions into the C-space map. These results show that, without optimisation, the implemented algorithm takes about 4 minutes and uses approximately 50 MB to compute a precise global C-space map for a 2-D six-degree-of-freedom revolute manipulator.

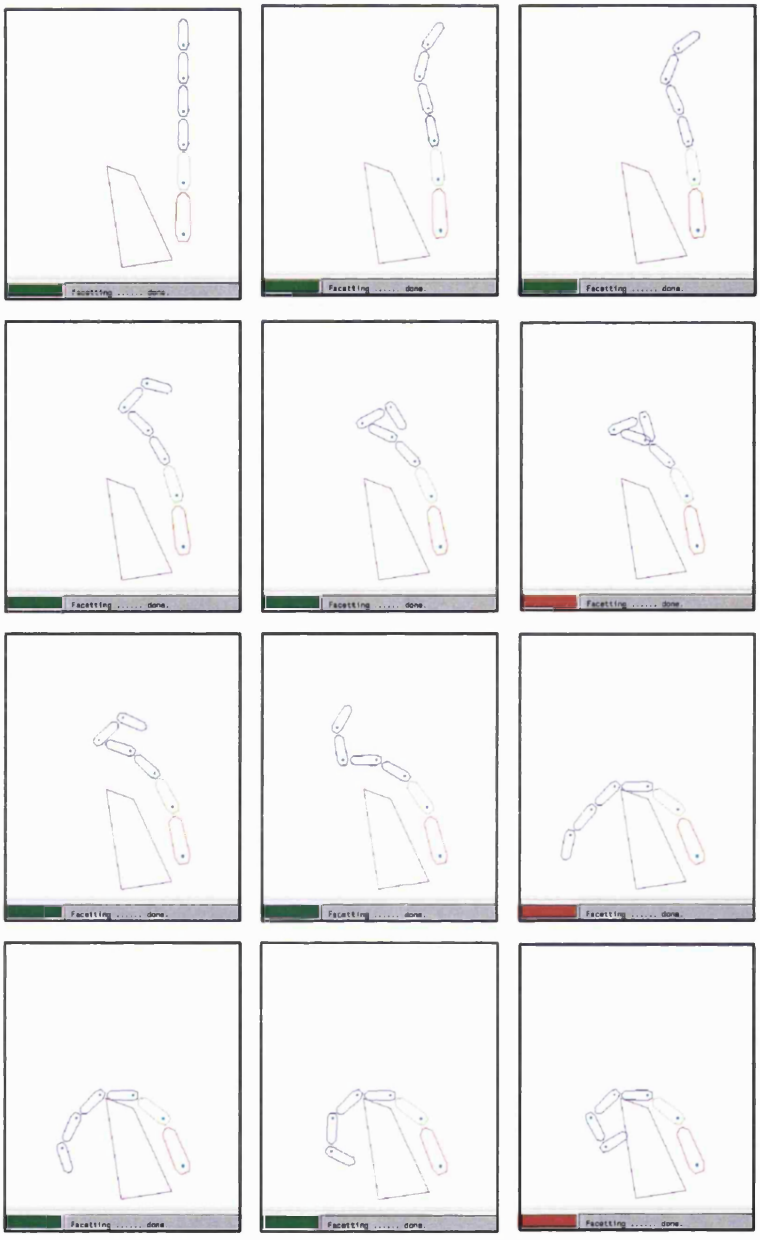


Figure 8.8: **Orienteer is used to explore the six-dimensional C-space map of a six-link 2-D revolute manipulator.**

8.6 Applying the omnimodel approach to a manipulator

As mentioned earlier, the omnimodel approach of introducing an additional dimension for each degree of freedom can be used for multiple nomad problems; it could also be used to model a manipulator by introducing the sweeps described in

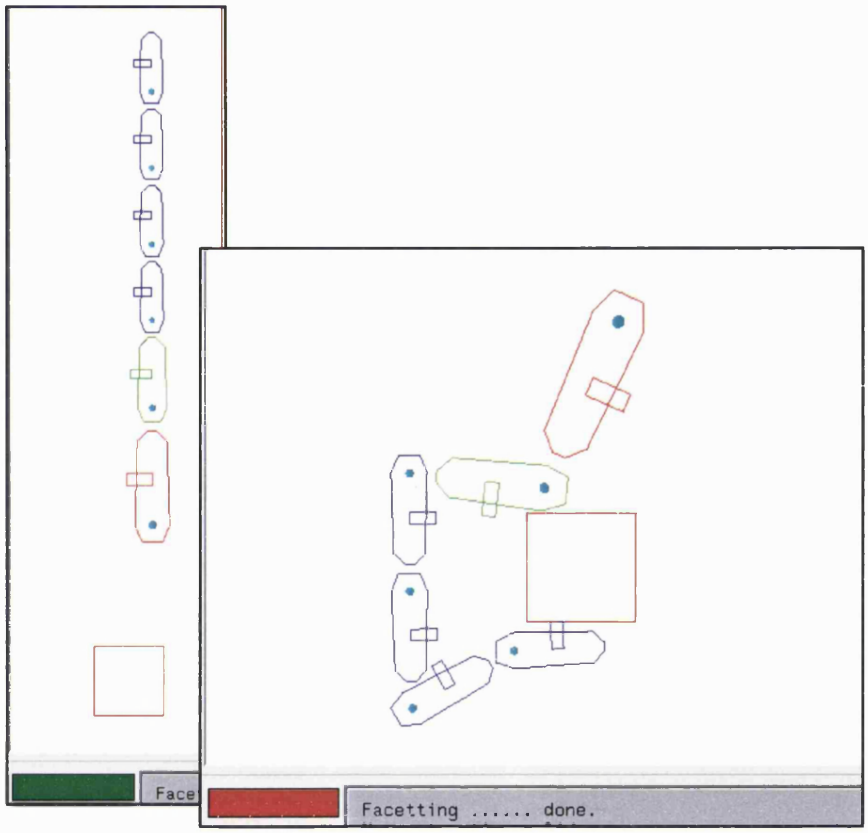


Figure 8.9: **A six-link 2-D revolute manipulator with non-convex links**

the above algorithm. In practice, however the large dimensionality would make recursive subdivision of the omnimodel intractable for all but the most simple problems. For that reason it has not been implemented.

8.7 Summary & conclusions

Multidimensional set-theoretic modelling offers three methods to map the C-space for a robotic manipulator. The most efficient of these adapts the analytical algorithms described in earlier chapters by effectively keeping each link static and sweeping the world around it. An implementation of this method has produced, for polygonal environments, complete and precise C-space maps of higher dimensionality than any of those found in the existing literature. The manipulator algorithm can handle unbounded sets, non-convex objects and three-dimensional cases, with the same strengths and weaknesses as the single-nomad algorithms.

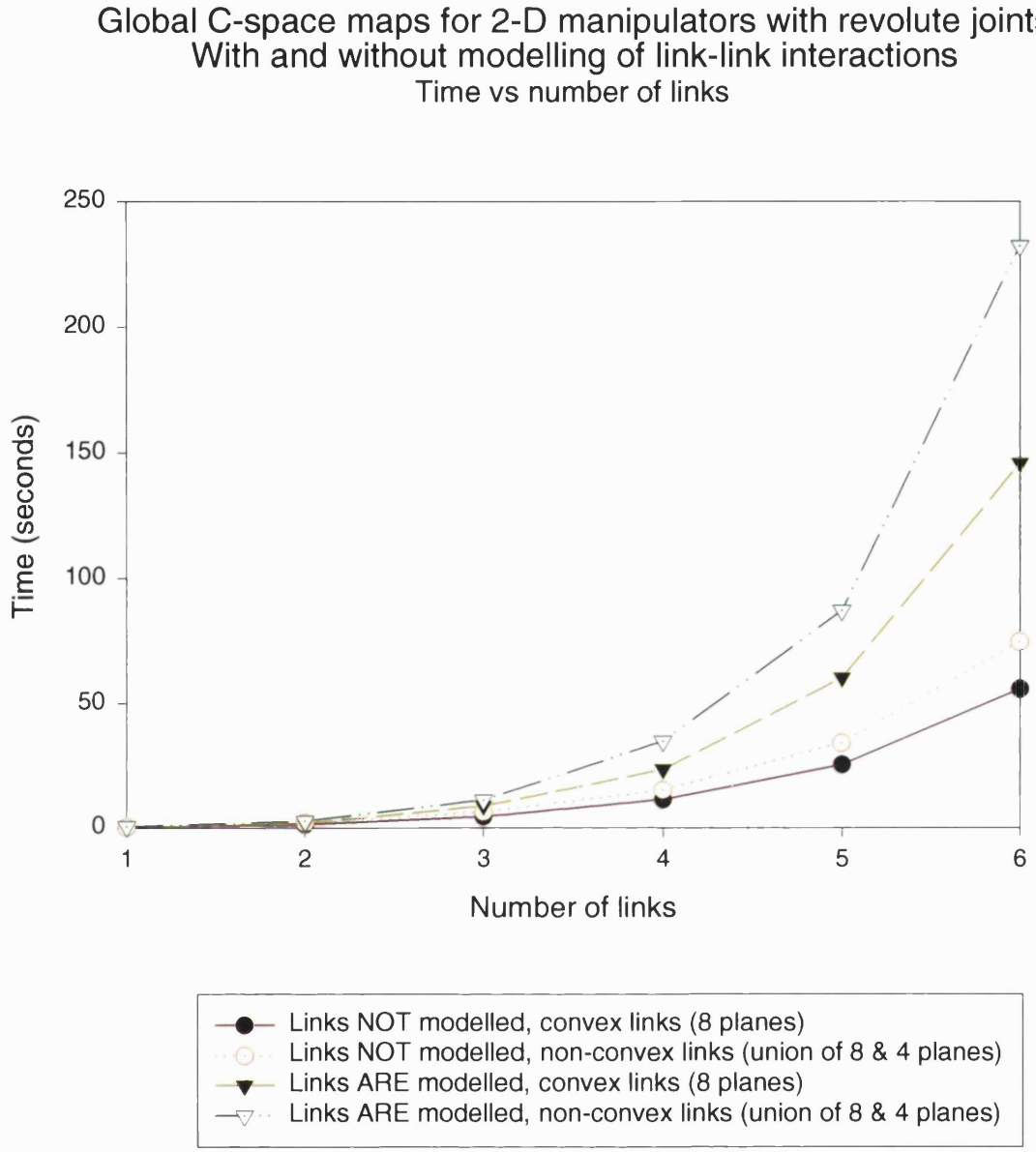


Figure 8.10: **Time taken plotted against the number of links for convex and non-convex links, both with and without incorporating link-link interactions**

Partly due to a limitation of our implementation of rotations, the algorithm has currently only been implemented for manipulators with revolute joints which rotate about the **z** axis. However in principle the algorithm itself extends to additional joint types in a straightforward manner—this is discussed in Chapter 9, Future Work.

Chapter 9

Future work

9.1 Introduction

This thesis has provided a broad investigation into how multidimensional set-theortic modelling can be used to compute global C-space maps. The focus on breadth rather than depth has meant that some of the algorithms described require further tuning, while the expanse of C-space applications means that not all application areas have been touched upon. In short, this thesis has raised as many questions as it has answered.

This chapter highlights some key issues which have been touched upon during the current research and which require further investigation.

9.2 Implementing an alternative representation of rotation

Perhaps the most significant improvements to the existing algorithms could be obtained by using an alternative representation for rotational degrees of freedom in three-dimensional space.

The most restrictive shortcoming of the current parameterisation (Roll-Pitch-Roll Euler angles) is that pure rotation about the **x** or **y** axes cannot be represented efficiently. This could be overcome by switching to the *Roll-Pitch-Yaw* variant of Euler angles (where the parameters correspond to rotation about the **x**, **y** and **z** axes) and by allowing all three parameters to take the range $[0, 2\pi)$ (at the cost of introducing redundancy).

Alternatively, the quaternion parameterisation could be used, which uses four parameters (corresponding to a three-dimensional axis vector and an angle). Although the introduction of an additional dimension would introduce additional work if the C-space is to be divided, the quaternion scheme would avoid the need for sin and cos operations since contact constraints and applicability conditions could be formulated as semi-algebraic sets (Canny [15]). Also, the axis-and-angle representation would allow rotation about an arbitrary axis to be modelled directly as one additional dimension.

The geometry of an omnimodel which is constructed using the quaternion representation is even harder to imagine than the Euler-angle equivalent. However, the principles behind omnimodel construction and projection are independent of the parameterisation used, so the approach described in Chapter 5 would still hold. The precise effect of the alternative representation on the algorithms would be a fascinating area to investigate.

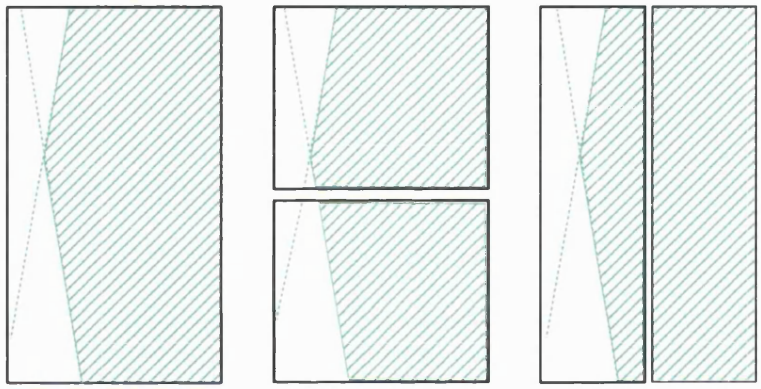
9.3 Improving omnimodel projection

The omnimodel projection algorithms described in Chapter 5 will never, of course, avoid the exponential time and memory requirements associated with increasing dimensionality. However, performance could be improved by fine-tuning in two key areas—the division strategy which determines how the omnimodel is chopped, and the resolution values which determine when division is terminated.

9.3.1 Division strategies which are more adaptive

All of the omnimodel projection algorithms described above are based on recursive division—either in the workspace dimensions, the C-space dimensions, or both. The versions described in this thesis are adaptive in that they adapt to the surface of the object being examined, but are blind in that when division is deemed necessary, the division plane is based soley on the box (specifically, the midpoint of the longest side) and not on the set contents.

As Figure 9.1 illustrates, a poor choice of division plane can be inefficient, unnecessarily replacing one model by two models of the same complexity as the first.



When model (a) is divided in the longest dimension, neither of the submodels (b) are any simpler than the parent – so nothing has been gained. In contrast, dividing in the other dimension resuls in one of the submodels being simplified to 'SOLID' which requires no further investigation.

Figure 9.1: **A poor choice of division plane is inefficient**

One approach to minimising this waste is to examine the range of surface normals exhibited by a set within a box, and to chop in the dimension which has the smallest range of normals. This test can be performed by substituting the box's intervals into the expressions obtained by partially differentiating the sets which respect to each of the dimensions. A second approach is to try more than one possible division plane, measure the effectiveness of each choice by pruning the contents to the potential submodels, then choose the best one before proceeding. It would be interesting to investigate if and when the additional cost of such look-ahead is justified.

The relationship between workspace, translation and rotation resolutions

As discussed in Section 5.5.1, the optimal relationship between resolutions in the workspace, translation and rotation dimensions is non-trivial to determine and requires further investigation. For example, given a target rotational resolution, what value should the workspace resolution be fixed at? Since translational C-space is isomorphic to the workspace (the property which enables a solution using Minkowski operations), an obvious default for the resolution in translational dimensions is an equivalence to the workspace resolution. However, this default is not necessarily optimal—dividing to a finer resolution in the workspace enables more leaf C-space cells to be classified as *safe* or *prohibited* so the thickness of the *contact* region is reduced.

9.3.2 Increasing the scope of analytical mapmaking

Chapters 6–8 show how I have exploited properties of set-theoretic geometric to produce precise global C-space maps of up to 18 dimensions for several application areas. This work could be extended in several directions.

Implementation of all DOF combinations

Using the techniques described in this thesis, algorithms have been implemented to compute precise global C-space maps for a wide range of cases. However, not every permutation of degrees of freedom has been implemented—for example, multiple nomad cases assume that all nomads have the same types of degrees of freedom.

Multiple nomads with full 3-D rotation

A key step in the algorithm described in Section 7.3 is to sweep the C-space obstacle which one nomad causes to another in order to incorporate the relative

orientation of the second nomad to the first. With one rotational degree of freedom each, the relative orientation is simply the difference between the rotation parameters for the two nomads; with full rotational freedom in three-dimensions, the relative transformation is obtained via manipulation of their rotation matrices. Although tractable, this has not yet been implemented so multiple nomad cases are restricted to one rotational degree of freedom per nomad.

Additional manipulator link types

The algorithm described in Chapter 8 demonstrates that multidimensional set-theoretic geometric modelling offers an elegant solution to computing precise global C-space maps for manipulators. In particular, the algorithm shows that, by making effort to keep the interesting link still and 'moving the world around it', redundant dimensions can be sliced away cleanly. The implementation of this algorithm shows that this approach works for two- and three-dimensional manipulators which have revolute joints, all of which rotate in the **xy** plane.

I hypothesise that the principles demonstrated in the algorithm extend to arbitrary joint types and that other types could be introduced as follows:

1. Spherical joints could be handled once relative three-dimensional orientations are formulated, as discussed in the previous point.
2. Revolute joints in other planes could also be introduced once relative three-dimensional orientations are implemented. However, the limitation of Roll-Pitch-Roll Euler angles mean that a more efficient implementation could be obtained once an alternative representation of rotation is introduced, as discussed above.
3. Prismatic joints should be straightforward to introduce to the current implementation, by applying the principles with translational degrees of freedom instead of rotational.

Mapmaking for multiple manipulators

By combining the principles of Chapter 7 and 8, it should be a straightforward matter to compute precise global C-space maps for cases involving multiple manipulators sharing a workspace.

Analytical solutions for semi-algebraic workspaces

In Section 6.8 I pointed out that Latombe [53, p. 149] proposes that if obstacle A and nomad B are represented as semi-algebraic sets, the C-space obstacle that A causes to B can be formulated as a Tarski sentence—it is therefore a semi-algebraic subset of the configuration space. This suggests that, in theory, a precise configuration-space map can be obtained using the set-theoretic representation for problems involving practically arbitrary geometry.

9.4 A hybrid mapmaker

This thesis demonstrates two very different ways multidimensional set-theoretic modelling can be used to obtain global C-space maps: The heuristic framework offered by omnimodel construction and projection is inherently general, but also inherently computationally expensive; analytical techniques allow a precise representation to be obtained quickly and compactly, even for cases with a highly dimensional C-space, but the method is only applicable to polygons and polyhedra, and since convex decomposition is required, it is only tractable for a subset of cases.

One goal of the research described here is to enable a novel hybrid mapmaker which combines the two approaches in a divide-and-conquer solution. In the proposed algorithm, an omnimodel would be constructed and then recursively divided until subomnimodels are reached which are simple enough for the analytical mapmaker to be applied; all the results of the subproblems would then be combined to solve the complete problem. In its purest form, the hybrid algorithm would use one of the algorithms described in Chapter 5 for all parts of the

omnimodel which represented interactions between curved geometry. However, since the projection algorithms introduce a *contact* region, an equally accurate C-space map might be obtained by applying analytical solutions in all cases—by approximating the objects by facetted versions as soon a tractable number of facets (and convex pieces) could be fitted to the objects. Division of the omnimodel would also provide an approach to decomposing problems into interactions between convex objects.

In summary, set-theoretic modelling would provide a flexible framework for investigating the trade-off between dividing and conquering. This fascinating avenue is left for future work.

9.5 Mapmaking for a dynamic environment

One property of the configuration space approach to spatial planning is that a C-space map itself is independent of time; constraints upon the velocity or acceleration of an object can be ignored even during the path planning stage (although non-holonomic constraints such as the turning arc on a car-like vehicle must be considered). However, as discussed in my survey of C-space applications beyond a single robot in a static environment Wise [106], an extended version of C-space, which incorporates time as a dimension, can be used to plan paths in environments where obstacles move in a predictable manner.

Introducing a time dimension would be a marked deviation from the existing algorithms and specific characteristics of time would require special attention—for example, trajectories traced through the C-space become restricted by velocity and acceleration constraints. However, the computation of configuration-time-space maps should be straightforward.

Consider, for example, a two-dimensional model which translates such that at time $\mathbf{t}$ its reference point has an $\mathbf{x}$ coordinate given by $f(\mathbf{x})$.

A three-dimensional model representing that object over the period of time $[\mathbf{t}_{lo}, \mathbf{t}_{hi}]$ is obtained by transforming the model in the same way described in Section 5.3.1, except that

1. The $\mathbf{x}'$ term is replaced by the tree for $f(\mathbf{x})$
2. The omnimodel box has the interval $[\mathbf{t}_0, \mathbf{t}_1]$ instead $[\mathbf{x}'_{lo}, \mathbf{x}'_{hi}]$

Extending this approach, a four-dimensional model could be constructed to represent a three-dimensional object with planned motion in six degrees of freedom—each degree of freedom would be parameterised in terms of $\mathbf{t}$, and the expression would be placed into the expanded tree instead of the dimension representing that degree of freedom.

Remember that the sweeps performed during omnimodel construction are the same ones used by the analytical mapmakers described in Chapters 6–8—so configuration-time-space maps could be produced by either the omnimodel projection algorithms or, more likely, analytical methods.

9.6 Using the C-space maps produced

This thesis has focussed specifically on the task of computing global C-space maps. Clearly, this is only a worthwhile task if the maps produced can be usefully employed. Here I outline two applications of the maps—analysis of behaviour using ray-tracing, and path planning.

9.6.1 C-space map analysis using ray-tracing

The current implementation of Orienteer enables C-space maps to be explored via a series of membership tests for discrete configurations. Although this is usually sufficient to validate a C-space map, very thin C-space obstacles can be 'jumped over' if the step length is not sufficiently small. An important improvement, then, is the introduction of *ray tracing* such that the path is checked between each start and end configuration. Multidimensional ray-tracing has been implemented in svLis-m, so its introduction into Orienteer is a minor task.

A more significant challenge for future work is to investigate how ray-tracing could be used to automate the design validation process. The starting point for

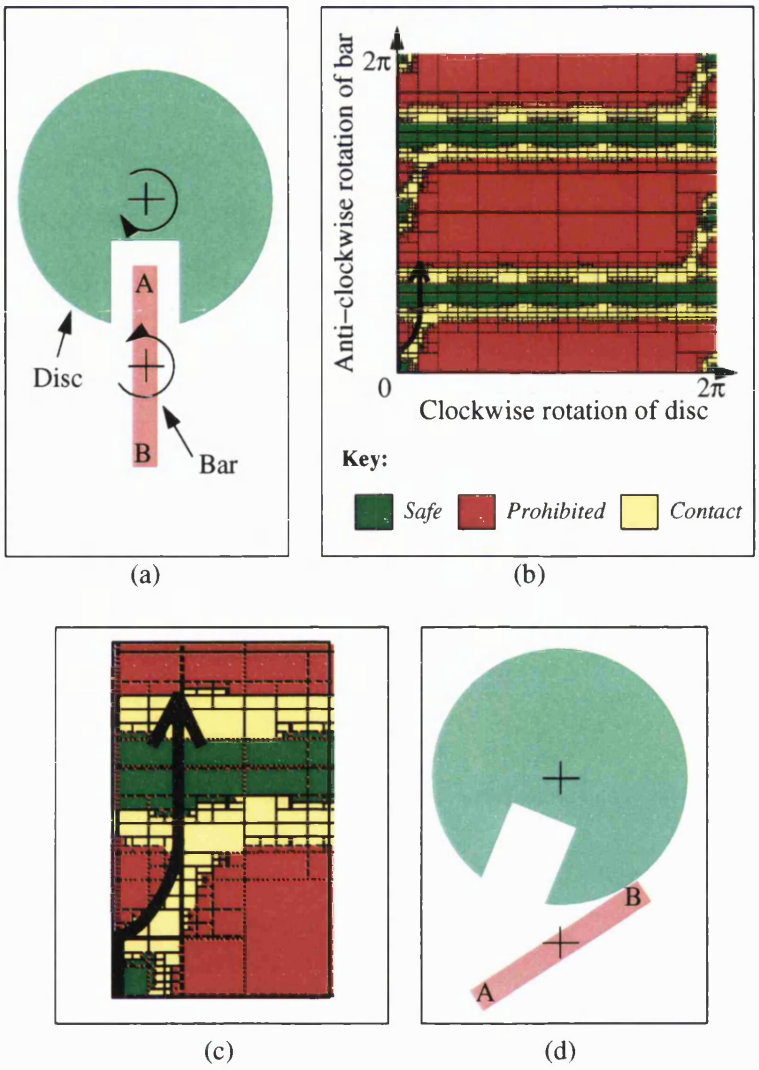
this investigation is that an input motion to a mechanism corresponds to a ray fired in to the C-space, and the mechanism's behaviour given that input motion can be predicted by tracing that ray as it passes through safe configurations and slides along contact surfaces. To illustrate this, consider the simple mechanism shown in Figure 9.2 (a), with a driving motion of an anti-clockwise rotation of the bar.

Figure 9.2 (b) shows the rotation as a ray tracing through the C-space map of the mechanism—Figure 9.2 (c) 'zooms in' to show the ray 'sliding' along the contact surface. Note that the ray can continue its upward motion if and only if it drifts to the right, corresponding to a clockwise rotation of the disc. After the disc has rotated approximately $\pi/4$, the bar disengages from the disc and the ray can continue vertically through a *safe* region of the C-space. However, at the other side of the *safe* region, the ray strikes a surface which is perpendicular to it, so the motion halts. This corresponds to the configuration shown in Figure 9.2 (d). The failure of the ray to continue through the C-space map indefinitely whilst periodically rotating the disc, indicates that the mechanism does not function as an indexing mechanism like a Geneva wheel.

9.6.2 Path planning

As highlighted in Chapter 1, the most common motivation for computing a C-space map is that it reduces path planning for an object (such as a robot) to a search for a line through the C-space map which connects start and goal configurations without entering *prohibited* regions. The maps produced by the algorithms described in this thesis offer two approaches to path-planning:

Optimal planning via a graph-search The binary spatial trees produced by the omnimodel projection algorithms discretise the C-space into connected cells. The same divided structure can be achieved by recursive subdivision of the precise C-space maps produced by the analytical methods. In either case, once neighbour information has been collated for each cell, a graph search algorithm such as the A* algorithm (Hart, Nilsson and Raphael 1968 [37]) can be used to find an optimal path for a specified criterion such as minimum distance. Path planning in this way is complete in that if a safe path through the discretisation



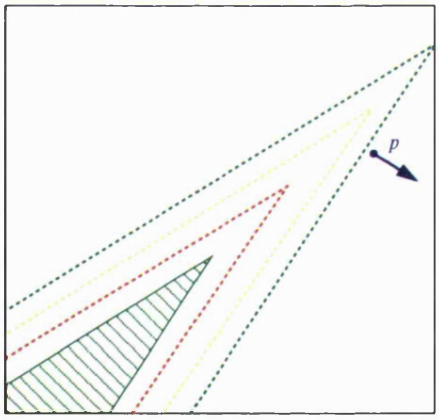
(a) An example two–dimensional mechanism – each object has one rotational degree of freedom.
(b) The C–space map for the mechanism in (a), computed using an omnimodel projection algorithm.
(c) A magnified illustration of the path traced through the C–space as the bar rotates anti–clockwise from the initial configuration
(d) The deadlock configuration which results when the bar impinges upon the disc

Figure 9.2: **The behaviour of a mechanism corresponds to a path traced through the C-space**

exists, it will be found. However, the algorithmic complexity of such exhaustive searches means they are only practical for C-spaces of low dimensionality (up to four dimensions).

Path planning in high-dimensional C-spaces using probabilistic roadmaps
One method for path planning in high-dimensional spaces is to construct and search a *probabilistic roadmap* [49]. This technique combines global and local search techniques by establishing a number of safe configurations, randomly distributed throughout the C-space, and then connecting these configurations by a network of safe paths by using local search techniques (such a potential field method) between those configurations.

Now, remember that the C-space maps produced by the analytical techniques of Chapters 6–8 are set-theoretic models of the same form as the input. As described in Chapter 3, the set-theoretic representation is effectively based on a potential field which is zero on the surface of the objects, negative inside the objects, and positive outside. In its original form, this potential field is not ideal for path planning—for example, since intersection is implemented by taking the maximum of two distance functions, the potential field is distorted around vertices even when objects are defined by linear halfspaces (Figure 9.3). Although there would be difficulties to overcome, an investigation of potential field techniques using multidimensional set-theoretic modelling is a potentially fruitful area for future work.



The dotted lines illustrate iso–values with respect to the potential field which defines solid region. Potential–field approaches to path–planning typically mimick a repulsive force from the obstacles – in this case, the potential field at point p is an inappropriate repulsion.

Figure 9.3: **The potential field around an object is distorted by intersections**

9.7 Incorporating parameterised models

Consider the simple two-dimensional model illustrated in Figure 9.4 (a), which is defined by the set-theoretic tree illustrated in Figure 9.4 (b).

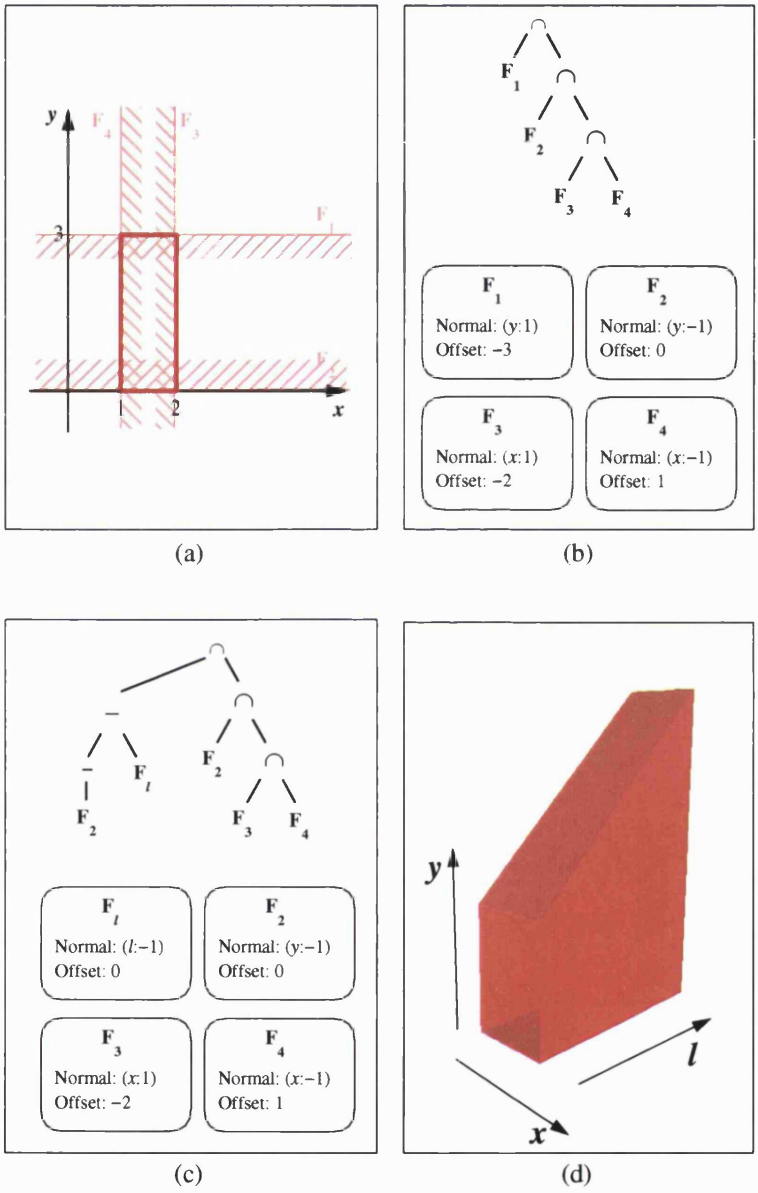


Figure 9.4: **A simple parameterised model**

Now consider the introduction of a parameter, l, corresponding to the length of the object in the **y** dimension. A parameterised version of the part can be represented by the set-theoretic tree in Figure 9.4 (c). If l is bound to some interval $[l_{lo}, l_{hi}]$, the result is a three dimensional model like that shown in Figure 9.4 (d),

which encapsulates all the versions of the part which have a length in y between the two bounds.

In [75], Parry-Barwick used parametric models represented in this way in conjunction with the omnimodel method described in Chapter 5 to perform feature recognition which identified not only where a parametrically defined part matched a template, but also what value of the parameter gave the best fit.

The same approach could be applied to the C-space algorithms described in this thesis. Instead of modelling the interactions between objects of fixed geometry, additional dimensions could be introduced to represent design parameters. The maps produced would be parameterised in both degrees of freedom and those design parameters, such that they could be sliced at any value of the design parameters to give the C-space map for that specific design.

9.8 A mechanism design tool

In [107], Adrian Bowyer, David Eisenthal and I describe the long term goal of this research—a mechanism analysis tool which facilitates mechanism design by calculating which design parameters provide the best mechanical performance.

Figure 9.5 illustrates how the proposed system would integrate several applications of multidimensional set-theoretic modelling, using configuration space as the common framework to encapsulate mechanism behaviour. The user would provide two inputs to the system: set-theoretic models of the components of a mechanism, and hints as to explicit constraints on the parts (for example, if two components are connected by a rotational joint, they can be labelled as having one rotational degree of freedom relative to each other, instead of treating each part and the connecting pin as having six degrees of freedom each). Feature recognition, an application of multidimensional set-theoretic modelling currently being investigated by Eisenthal [27], would then be used to spot further explicit constraints imposed by features of the components (such as a mating tongue and groove). Both sets of constraints (user-supplied and automatically-detected) would then be fed to a constraint modelling system—another application of multidimensional set-theoretic modelling being investigated by Eisenthal [27]. In

addition to constraints imposed by lower kinematic pairs such as joints and mating features, the behaviour of the parts is restricted by their shape and the fact they cannot overlap; these constraints would be modelled by the C-space mapping algorithms described in this thesis.

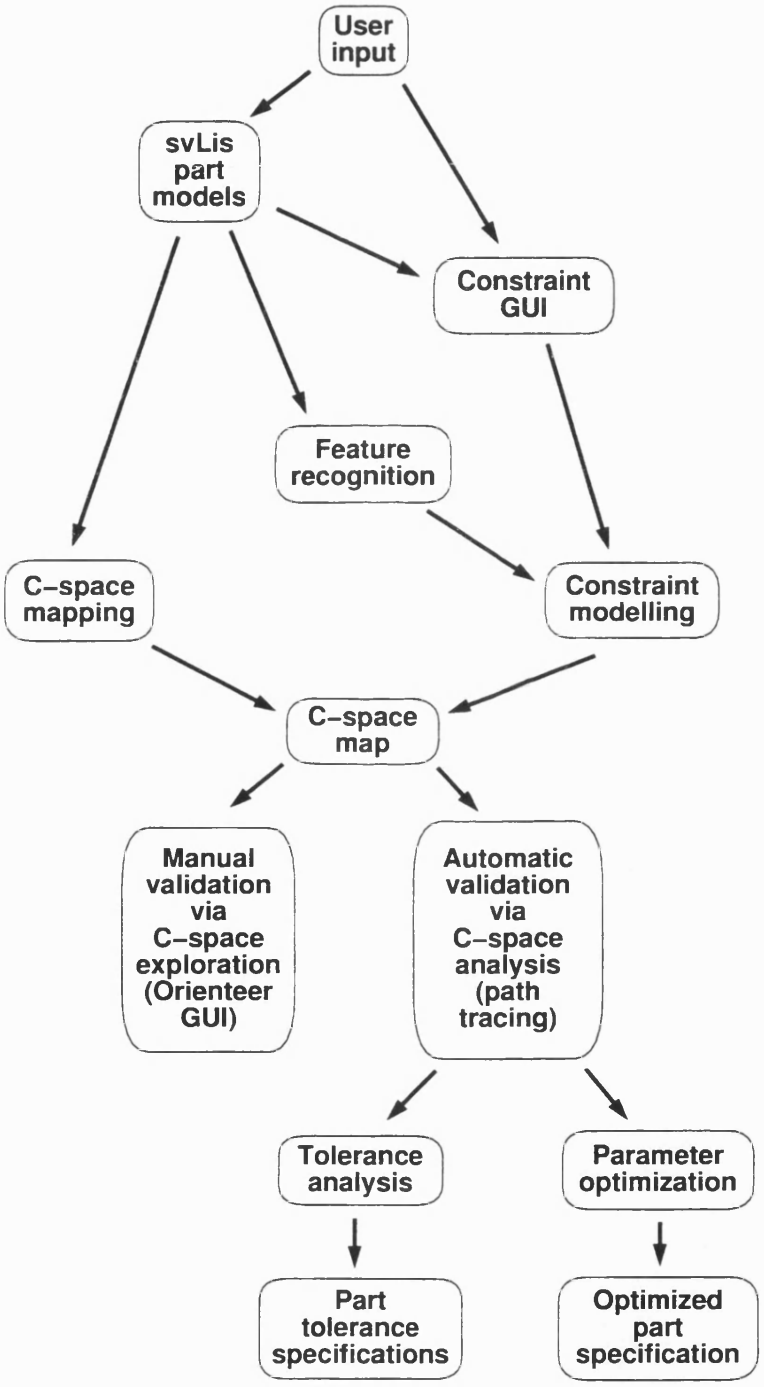


Figure 9.5: **A schematic for a mechanism design tool based on integrating applications of multidimensional set-theoretic modelling**

Constraint modelling and C-space mapping both represent the interactions between parts as a geometric model embedded in the configuration space. The C-space map obtained by combining the results would encapsulate the kinematic behaviour of the mechanism, and this could be analysed to aid the design process in several ways. The first, which is already possible using Orienteer, would be to explore the C-space in a graphical user interface to observe if the desired mechanism behaviour can be realised by the specified geometry of the parts. 'Virtual prototyping' in this way could be taken one step further if a simple model of dynamics is introduced to the simulation. In [85], Sacks and Joskowicz establish such a model, which makes simplifications such as:

- Forces impart a constant linear or angular velocity on a component along a fixed translational axis or around a rotational axis.
- There is no inertia—the velocity drops to zero the instant the force stops acting.
- Collisions among parts are perfectly inelastic.
- Falling objects instantaneously reach terminal velocity.
- A spring with one end fixed is modelled as a constant force and the free end never oscillates.
- Friction constrains the relative motion of touching parts; every surface has a coefficient of friction of zero (smooth) or one (sticky); friction acts solely between touching pairs of sticky faces.

The encapsulation of kinematic behaviour provided by a C-space map potentially enables a more automated method of design validation. Since a specific behaviour of a mechanism corresponds to a path through the C-space, and paths through a C-space can be predicted via ray-tracing (Section 9.6.1), there is potential for automatically identifying if a specific design gives a desired behaviour by testing if the desired path is traced.

Further possibilities are introduced if the models provided to the mechanism analysis tool are parameterised—ultimately a *configuration-design-space* map, which has dimensions corresponding to both degrees of freedom and design parameters, could be used to automatically determine what range of parameters define a

working mechanism. Going even further than identifying viable parameters, the analysis system could compute an *optimal* design using criteria such as maximum average tolerance, and minimum total volume of material. Although this is an ambitious target, some insights into how this might be achieved are provided in [107].

9.9 Summary & conclusions

The research described in this thesis has a highlighted a wide range of avenues which might be fruitfully investigated in the future. Continuing the focus on the computation of global C-space maps, the short term priorities are to:

- Implement the quaternion parameterisation of three-dimensional rotation, which would enable efficient treatment of rotation about an arbitrary axis and would enable omnimodels and precise contact surfaces to be represented as semi-algebraic sets.
- Increase the scope of the analytical methods by
  - Increasing the types of manipulator joint types handled
  - Extending the current algorithms to cases involving dynamic obstacles and multiple manipulators
  - Formulating precise contact constraints and applicability conditions for some curved objects, such as spheres and cylinders
- Integrate the omnimodel and analytical approaches as a hybrid divide-and-conquer algorithm

Changing focus to applications of C-space techniques, the maps produced by the algorithms described in this thesis have properties which could be exploited for path planning, mechanism analysis and ultimately, with the introduction of parametric models, mechanism design optimisation.

Chapter 10

Summary & conclusions

C-space mapmaking is an ongoing area of research, driven by the wide range of applications and the challenge of increasing the dimensionality of the maps and/or the range of allowable geometry accepted as input. As part of that research, this thesis investigates the computation of global configuration-space maps using multidimensional set-theoretic modelling. Specifically, I apply a set-theoretic geometric modelling approach which employs the *linear halfspace basis* (representing all primitives as expressions in terms of linear halfspaces) and makes extensive use of *interval arithmetic*. Since both of these extend in a straightforward manner to any number of dimensions, my colleagues and I have developed svLis-m—a set-theoretic modelling kernel which can represent objects of near-arbitrary dimensionality.

I describe two ways svLis-m can be used to compute global C-space maps. The first combines workspace dimensions with degrees of freedom to construct a static model which represents every interaction that occurs as a nomad (or system of nomads) exercises its degrees of freedom. When the interference regions within this model are projected into the configuration space, they map the C-space obstacles to the nomad or system of nomads. I describe five heuristic algorithms which perform this projection to obtain an approximate map of the C-space. All have been successfully implemented, and they are inherently general. However, they are also inherently computationally expensive so they are not appropriate for computing highly dimensional C-spaces to a practical resolution.

The second approach to computing global C-space maps using svLis-m is to formulate precise analytical representations for the contact surfaces which result when polygons or polyhedra interact. Set-theoretic modelling has several properties which suit it to this task:

- The strong relationship between convexity and intersection makes it straightforward to compute the C-space obstacle caused by one convex object to another.
- The implicit property of set-theoretic modelling makes it unnecessary to calculate the intersection between contact constraints and applicability conditions, or to compute neighbourhood information.
- The dimension-independent notation enables multidimensional C-space obstacles to be computed using straightforward extensions to the algorithms for designed for simpler cases.
- All of the contact constraints and applicability conditions can be formulated by expanding elements of the rigid body matrix and expressing the result as a tree in the linear halfspace basis (this approach is also used to compute the omnimodels mentioned above).

By exploiting these properties, I have implemented a precise global C-space mapmaker which can handle three-dimensional objects with a full six degrees of freedom. Moreover, I have shown that these analytical algorithms can be extended to incorporate multiple independent nomads and then to manipulators. These extension have also been implemented, resulting in validated precise C-space maps of up to eighteen dimensions.

The focus on breadth rather than depth has meant that this thesis has had to leave many avenues for future work. I suggest that the following would be particularly interesting and potentially fruitful:

- An implementation of the quaternion parameterisation of three-dimensional rotation, as an alternative to the current Roll-Pitch-Roll Euler angles, which have several shortcomings.
- Extensions to the analytical methods to encompass

  - Additional manipulator joint types
  - Dynamic obstacles and multiple manipulators
  - Some curved objects, such as spheres and cylinders
- An integration of the omnimodel and analytical approaches as a hybrid divide-and-conquer algorithm.

If you would like to get in touch to discuss any issues raised in this thesis, please e-mail `kevindwise@hotmail.com`. Thanks for your attention.

Appendix A

Implementation of a rotational sweep

The C++ code below is the svLis-m implementation of a rotational sweep. As described in Section 5.3.2, this creates a six-dimensional helicoid set representing the original flat as it exercises three rotational degrees of freedom (parameterised by Roll-Pitch-Roll Euler angles).

Each of the sets sent in (`x`, `y`, `z`, `phi`, `theta` and `psi`) is a 1-D flat with a normal pointing in the dimension identified.

```
SvmSet wdkEulerRotate(
    const SvmFlat& orig,
    const SvmSet& phi,
    const SvmSet& theta,
    const SvmSet& psi,
    const SvmSet& x,
    const SvmSet& y,
    const SvmSet& z
)
{
    SvmReal x_init = orig.normal().val(x.space().dim(0));
    SvmReal y_init = orig.normal().val(y.space().dim(0));
    SvmReal z_init = orig.normal().val(z.space().dim(0));

    // Temps used to build the tree.
    SvmSet temp_x, temp_y, temp_z;

    SvmSet x_coeff = x_init;
    SvmSet y_coeff = y_init;
    SvmSet z_coeff = z_init;

    SvmSet cos_phi = svmCos(phi);
    SvmSet sin_phi = svmSin(phi);
    SvmSet cos_theta = svmCos(theta);
    SvmSet sin_theta = svmSin(theta);
    SvmSet cos_psi = svmCos(psi);
    SvmSet sin_psi = svmSin(psi);

    // Rotate by psi about z.
    temp_x = (x_coeff * cos_psi) + (y_coeff *-sin_psi);
    temp_y = (x_coeff * sin_psi) + (y_coeff * cos_psi);

    x_coeff = temp_x;
    y_coeff = temp_y;

    // Postmultiply by theta about y.
    temp_x = (x_coeff * cos_theta) + (z_coeff * sin_theta);
    temp_z = (x_coeff *-sin_theta) + (z_coeff * cos_theta);

    x_coeff = temp_x;
    z_coeff = temp_z;

    // Postmultiply by phi about z.
    temp_x = (x_coeff * cos_phi) + (y_coeff *-sin_phi);
    temp_y = (x_coeff * sin_phi) + (y_coeff * cos_phi);

    x_coeff = temp_x;
    y_coeff = temp_y;
    z_coeff = temp_z;

    SvmSet result = x_coeff*x + y_coeff*y + z_coeff*z + orig.distance();

    return (result);
}
```

Appendix B

Tables of data for Section 5.12

Resolution	collDetPoint	collDetBox	collDetBSD	isotropicBSD	L-collDetBSD
4	1	2	4	1	2
5	8	10	15	7	10
6	38	52	62	34	37
7	181	242	225	164	124
8	808	1032	781	653	424
9	3634	4781	2727	2553	1501

Table B.1: **Time taken (s.) vs. resolution for the 2-D 'Medium spheres' case with 2 DOFs.**

Resolution	collDetPoint	collDetBox	collDetBSD	isotropicBSD	L-collDetBSD
3	0	1	2	1	1
4	4	6	12	8	8
5	25	34	56	34	34
6	125	177	194	171	114
7	557	773	699	670	402
8	2322	3198	2378	2611	1449
9	10935	14164	7834	10579	4752

Table B.2: **Time taken (s.) vs. resolution for the 2-D 'Big spheres' case with 2 DOFs.**

Resolution	collDetPoint	collDetBox	collDetBSD	isotropicBSD	L-collDetBSD
3	18	23	33	11	18
4	217	322	571	225	332
5	2286	3960	6287	3011	3431
6	23407	40785	57278	32518	30193

Table B.3: **Time (on an SG origin) vs. resolution for the 3 3-D 'Med spheres' case with three translational DOFs**

References

[1] E. A. Abbot. *Flatland.* Alden Press, 1884.

[2] M. Adamowicz and A. Albano. Nesting two-dimensional shapes in rectangular modules. *Computer Aided Design*, 8(1):27, 1976.

[3] P. Adolphs and D. Nafziger. A method for fast computation of collision-free robot movements in configuration space. In *IEEE International Workshop on Intelligent Robots and systems, Tokyo, Japan, July 4–6*, pages 5–12, 1990.

[4] F. Avnaim and J. D. Boissonnat. Polygon placement under translation and rotation. *Springer's Lecture Notes in Computer Science Series*, 294:323–333, 1988.

[5] F. Avnaim, J. D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. *Springer's Lecture Notes in Computer Science Series*, 391:67–86, 1988.

[6] C. Bajaj and M. S. Kim. Generation of configuration space obstacles — the case of a moving sphere. *IEEE Journal of Robotics and Automation*, 4(1):94–99, 1988.

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

[8] C. Bellier, C. Laugier, E. Mazer, and J. Troccaz. Planning/executing six DOF robot motions in complex environments. *Proc 91 IEEE RSJ International Workshop Intelligence Robots Systems IROS 91*, pages 91–96, 1992.

[9] Jakob Berchtold and Adrian Bowyer. Bézier surfaces in set-theoretic geometric modelling. In *CSG98 Set-theoretic geometric modelling: techniques and applications*, pages 1–16. Information geometers, April 1998.

[10] A. Bowyer. *Svlis—Introduction and User Manual.* Information Geometers, second edition, 1995. www.bath.ac.uk/~ensab/G_mod/Svlis.

[11] M. S. Branicky and W. S. Newman. Rapid computation of configuration space obstacles. *Proc 1990 IEEE International Conference Robotics Automation*, pages 304–310, 1990.

[12] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems Man and Cybernetics*, 15(2):224–233, 1985.

[13] R. C. Brost. Computing metric and topological properties of configuration-space obstacles. *International Conference Robotics Automation 1989*, 1:170–176, 1989.

[14] J.M. Cameron and R. Culley. Determining the minimum translational distance between two convex polyhedra. In *IEEE Conf. Robotics and Automation, San Francisco, April 1986*, pages 591–596. IEEE, Piscataway, NJ, USA, 1986.

[15] J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, Mass., 1988.

[16] M. Cesati and H. T. Wareham. Parameterized complexity analysis in robot motion planning. In *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics. Part 1 (Of 5), Vancouver, BC, Can, Oct 22-25 1995, (Conf. Code 44222)*, volume 1, pages 880–885. IEEE, Piscataway, NJ, USA, 1995.

[17] R. H. T. Chan, P. K. S. Tam, and D. N. K. Leung. Solving the motion planning problem by using neural networks. *Robotica*, 12(4):323–333, 1994.

[18] P. C. Chen and Y. K. Hwang. SANDROS: a motion planner with performance proportional to task difficulty. *Proceedings — IEEE International Conference on Robotics and Automation*, 3:2346–2353, 1992.

[19] H. Cheng and H. D. Cheng. Feasible map algorithm for path planning. *Robotics and Autonomous Systems*, 17(3):149–170, 1996.

[20] H. S. M. Coxeter. *Introduction to Geometry.* John Wiley and Sons, 1963.

[21] B. Curto and V. Moreno. Mathematical formalism for the fast evaluation of the configuration space. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation, Cira, Monterey, CA, USA, Jul 10-11 1997, (Conf. Code 46973)Record - 85*, pages 194–199. IEEE, Piscataway, NJ, USA, 1997.

[22] F. Dehne, A. L. Hassenklover, and J. R. Sack. Computing the configuration space for a robot on a mesh-of-processors. *Parallel Computing*, 12(2):221–231, 1989.

[23] M. T. Depedro and R. G. Rosa. Robot path planning in the configuration space with automatic obstacle transformation. *Cybernetics and Systems*, 23(3–4):367–378, 1992.

[24] B. R. Donald. On motion planning with 6 degrees of freedom: Solving the intersection problems in configuration space. *Proc IEEE International Conference on Robotics and Automation*, pages 536–541, 1985.

[25] B. R. Donald. A search algorithm for motion planning with 6–degrees of freedom. *Artificial Intelligence*, 31(3):295–353, 1987.

[26] G. Duelen and C. Willnow. Path planning of transfer motions for industrial robots by heuristically controlled decomposition of the configuration space. In S. G. Tzafestas, editor, *Robotic Systems: Proceedings of the 1991 EURISCON Conference, Korfu*, pages 217–224. Kluwer Academic Publishers, 1991.

[27] D. C. R. Eisenthal. Applications of multidimensional set theoretic geometry. Technical Report 054/97, Dept. of Mechanical Engineering, University of Bath, December 1997.

[28] B. Faltings. Qualitative kinematics in mechanisms. In *Proceedings of IJCAI-87*, pages 436–442, 1987.

[29] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *Proceedings of the IEEE Int. Conf. Robotics, Atlanta, GA*, pages 504–512, March 1984.

[30] B. Faverjon and P. Tournassoud. Motion planning for manipulators in complex environments. *Geometry & Robotics, in Springers' Lecture Notes in Computer Science Series*, 391:87, 1988.

[31] K. Fujimura. Motion planning amidst dynamic obstacles in three dimensions. *1993 International Conference on Intelligent Robots and Systems*, page 1387, 1993.

[32] Q. Ge and J. M. McCarthy. Equations for boundaries of joint obstacles for planar robots. *International Conference Robotics Automation*, 1:164–169, 1989.

[33] Q. J. Ge and J. M. McCarthy. An algebraic formulation of configuration-space obstacles for spatial robots. *Proc 1990 IEEE International Conference Robotics Automation*, pages 1542–1547, 1990.

[34] PK Ghosh. A unified computational framework for minkowski operations. *Computing & Graphics*, 17(4):357–378, 1993.

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

[36] D. Halperin, M. H. Overmars, and M. Sharir. Efficient motion planning for an L-shaped object. *SIAM Journal on Computing*, 21(1):1–23, 1992.

[37] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions Systems, Science and Cybernetics*, SSC-4(2):100, 1968.

[38] T. Hasegawa and H. Terasaki. Collision avoidance — a divide-and-conquer approach by space characterization and intermediate goals. *IEEE Transactions on Systems Man and Cybernetics*, 18(3):337–347, 1988.

[39] J. H. Heegaard. Efficient parametrization of the configuration space for rigid multi-body contact and impact. In *Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exposition, Atlanta, GA, USA, Nov 17–22 1996, (Conf. Code 45867)*, volume 33, pages 431–432. ASME, New York, NY, USA, 1996.

[40] C. M. Hoffmann. *Geometric & Solid Modeling: an Introduction.* Morgan Kaufmann Publishers, Inc., 1989.

[41] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: P-space-hardness of the "warehouseman's problem". *International Journal of Robotics Research*, 3(4):76, 1984.

[42] Y. K. Hwang. Boundary equations of configuration obstacles for manipulators. *Proc 1990 IEEE International Conference Robotics Automation*, page 298, 1990.

[43] Y. K. Hwang. Current status and future research in motion planning. In *Proceedings, IEEE International Symposium on Assembly and Task Planning, Pittsburgh, PA, USA, 10–11 Aug. 1995*, pages 431–432, 1995.

[44] Y. K. Hwang and N. Ahuja. Gross motion planning — a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[45] A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. *IEEE — ?*, page 361, 1990.

[46] R. A. Jarvis. Growing polyhedral obstacles for planning collision-free paths. *Australian Computer Journal*, 15(3):103–110, 1983.

[47] A. Kaul and R. T. Farouki. Computing minkowski sums of plane-curves. *International Journal of Computational Geometry & Applications*, 5(4):413–432, 1995.

[48] L. Kavraki. Computation of configuration-space obstacles using the Fast Fourier Transform. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 3, pages 255–261, 1993.

[49] L. E. Kavraki. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[50] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proceedings of the 1996 IEEE 13Th International Conference on Robotics and Automation. Part 4 (Of 4), Minneapolis, MN, USA, Apr 22-28 1996, (Conf. Code 44943)*, volume 4, pages 3020–3025. IEEE, Piscataway, NJ, USA, 1996.

[51] M. G. Kendall. *A course in the geometry on n dimensions.* Griffin, 1961.

[52] M. Kohler and M. Spreng. Fast computation of the C-space of convex 2-d algebraic objects. *International Journal of Robotics Research*, 14(6):590–608, 1995.

[53] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publ., 1993.

[54] C. Laugier and F. Germain. An adaptative collision-free trajectory planner. *International Conference Advanced Robotics*, pages 33–41, 1985.

[55] J. P. Laumond. Obstacle growing in a nonpolygonal world. *Information Processing Letters*, 25(1):41–50, 1987.

[56] J. P. Laumond, T. Simeon, R. Chatila, and G. Giralt. Trajectory planning and motion control for mobile robots. *Geometry & Robotics, in Springers Lecture Notes in Computer Science Series*, 391:133, 1988.

[57] I. K. Lee, M. S. Kim, and G. Elber. Polynomial/rational approximation of minkowski sum boundary curves. Technical Report PIRL-TR-97-005, Computer Graphics Lab., PIRL, POSTECH, San 31, Hyoja Dong, Pohang 790–784, South Korea, June 1997.

[58] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics (ACM)*, 24(4):327–335, 1990.

[59] P. L. Lin and S. H. Chang. A shortest path algorithm for a nonrotating object among obstacles of arbitrary shapes. *IEEE Transactions Systems, Man and Cybernetics*, 23(3):825, 1993.

[60] Y. H. Liu and H. Onda. Constructing an approximate representation of a configuration space without using an intersection check. *1993 International Conference on Intelligent Robots and Systems*, pages 644–651, 1993.

[61] LogiCad. The magellan/SPACE MOUSE "classic". World-Wide Web, 1999. http://www.spacemouse.com/products/.

[62] T. Lozano-Pérez. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560, 1979.

[63] T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transactions on Computers*, 32(2):108–119, 1983.

[64] T. Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, 3(3):224–238, 1987.

[65] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man and Cybernetics*, 11(10):681, 81.

[66] C. Ma, W. Li, Y. Yang, and L. Chang. Robot motion planning with many degrees of freedom. In *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics. Part 1 (Of 5), Vancouver, BC, Can, Oct 22-25 1995, (Conf. Code 44222)*, volume 1, pages 892–897. IEEE, Piscataway, NJ, USA, 1995.

[67] A. A. Maciejewski and J. J. Fox. Path planning and the topology of configuration-space. *IEEE Transactions on Robotics and Automation*, 9(4):444–456, 1993.

[68] D. Marr and H. K. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. Technical Report AIM-416, Artificial Intell. Lab, MIT, Cambridge, Mass., May 1977.

[69] Ramon E. Moore. *Interval analysis.* Prentice Hall, 1966.

[70] T. Murakami and D. C. Gossard. Mechanism concept retrieval by behavioral specification using configuration space. *ASME Design Engineering Division (Publication): Design Theory and Methodology*, 42:343–350, 1992.

[71] F. N-Nagy and A. Siegler. *Engineering foundations of robotics.* Prentice-Hall, 1987.

[72] W. S. Newman and M. S. Branicky. Real-time configuration space transforms for obstacle avoidance. *International Journal of Robotics Research*, 10(6):650–667, 1991.

[73] B. Paden, A. Mees, and M. Fisher. Path planning using a Jacobian-based freespace generation algorithm. *Proceedings IEEE International Conference on Robotics and Automation*, pages 1732–1737, 1989.

[74] S. Parry-Barwick and A. Bowyer. Multidimensional set-theoretic feature recognition. *Computer-Aided Design*, 27(10):731–740, 1995.

[75] S. J. Parry-Barwick. *Multi-dimensional Set-theoretic Geometric Modelling.* PhD thesis, Dept. of Mechanical Engineering, University of Bath, 1995.

[76] D. Parsons and J. Canny. A motion planner for multiple mobile robots. *Proc 1990 IEEE International Conference Robotics Automation*, 1:8, 1990.

[77] R.P. Paul. *Robot Manipulators.* MIT press, 1982.

[78] E. Ralli and G. Hirzinger. Global and resolution complete path planner for up to 6dof robot manipulators. In *Proceedings of the 1996 IEEE 13Th International Conference on Robotics and Automation. Part 4 (Of 4), Minneapolis, MN, USA, Apr 22-28 1996, (Conf. Code 44943)Record - 26*, volume 4, pages 3295–3302. IEEE, Piscataway, NJ, USA, 1996.

[79] E. Ralli and G. Hirzinger. Efficient c-space modelling using kohonen maps. In *Proceedings of the 1997 8Th International Conference on Advanced Robotics, Icar'97, Monterey, CA, USA, Jul 7-9 1997, (Conf. Code 46955)Record - 74*, pages 433–438. IEEE, Piscataway, NJ, USA, 1997.

[80] W. E. Red and H. V. Truong-Cao. Configuration maps for robot path planning in two dimensions. *ASME Journal of Dynamic Systems, Measurement and Control*, 107(4):292–298, 1985.

[81] P. Regli, K. Lamboglia, G. Garreton, M. Neeracher, M. Westermann, N. Strecker, and W. Fichtner. Multidimensional geometric modeling for 3d tcad. *Microelectronic Engineering*, 34(1):101–115, 1996.

[82] J. H. Reif. Complexity of the mover's problem and generalizations. *Proceedings of the 20th Symposium on the Foundations of Computer Science*, pages 421–427, 1979.

[83] A. Requicha and H. Voelcker. Solid modeling: Current status and research directions. *IEEE Computer Graphics Applications*, pages 25–37, 1983.

[84] J.R. Rossignac. Processing disjunctive forms directly from CSG graphs. In *CSG 94 Set-theoretic Solid Modelling: Techniques and Applications*, volume 4, pages 55–70. Information Geometers, Winchester, 1994.

[85] E. Sacks and L. Joskowicz. Automated modeling and kinematic simulation of mechanisms. *Computer Aided Design*, 25(2):106–118, 1993.

[86] H. Samet. Spatial data structures. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 361–385. Addison Wesley/ACM Press, 1995.

[87] J. T. Schwartz and M. Sharir. On the piano movers' problem I: The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Comm on Pure & Applied Maths*, 36:345, 1983.

[88] A. Schweikard and R.H. Wilson. Assembly sequences for polyhedra. *Algorithmica*, 13:539–552, 1995.

[89] M. Selig. *Introductory robotics.* Prentice-Hall, 1992.

[90] Z. Shiller and Y. R. Gwo. Collision-free path planning of articulated manipulators. *Journal of Mechanical Design*, 115(4):901–908, 1993.

[91] T. Siméon. Planning collision-free trajectories by a configuration space approach. *Geometry & Robotics, in Springers Lecture Notes in Computer Science Series*, 391:116–132, 1988.

[92] W.E. Snyder. *Industrial Robots.* Prentice-Hall, 1985.

[93] J. Solano Gonzalez and D. I. Jones. Parallel computation of configuration space. *Robotica*, 14(pt2):205–212, 1996.

[94] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley & Sons, 1989.

[95] D. Subramanian and C. S. E. Wang. Kinematic synthesis with configuration-spaces. *Research in Engineering Design-Theory Applications and Concurrent Engineering*, 7(3):193–213, 1995.

[96] L. Toft. *Definitions and formulae for students.* Pitman, 1951.

[97] S. K. Tso and K. P. Liu. Fast motion planner based on configuration space. *1993 International Conference Intelligence Robotics Systems*, pages 1401–1408, 1993.

[98] S. M. Udupa. Collision detection and avoidance in computer controlled manipulators. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 737–748, 1977.

[99] G. Vaněček, Jr. Brep-index: A multidimensional space partitioning tree. In J. Rossignac and J. Turner, editors, *ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications (Austin, Tex.*, pages 35–44. ACM Press, New York, 1991.

[100] B. J. H. Verwer. A multiresolution work space, multiresolution configuration space approach to solve the path planning problem. *Proc 1990 IEEE International Conference Robotics Automation*, pages 2107–2112, 1990.

[101] I. D. Voiculescu. Motion planning with six degrees of freedom. Technical Report TR-32/98, Department of Mechanical Engineering, Faculty of Engineering and Design, University of Bath, 1998.

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

[103] K. D. Wise. Generalized comparison of bintree and 2^n-tree storage requirements. Technical Report 053/97, Dept. of Mechanical Engineering, University of Bath, December 1997.

[104] K. D. Wise and A. Bowyer. A survey of global configuration-space mapping techniques for a single robot in a static environment. *International Journal of Robotics Research.* Accepted. Awaiting publication.

[105] K. D. Wise and A. Bowyer. Using CSG models in many dimensions to map where things can and cannot go. In *Proceedings from CSG 96 Set-theoretic Solid Modelling: Techniques and Applications*, pages 359–376, 1996.

[106] K. D. Wise and A. Bowyer. Configuration-space mapping for multiple moving objects: a survey of techniques and applications. Technical Report 09/98, Dept. of Mechanical Engineering, University of Bath, February 1998.

[107] K. D. Wise, D. Eisenthal, and A. Bowyer. Design optimization of rigid-body mechanisms via multidimensional CSG, 1998. Presented at the 1997 ASME Design For Manufacturing Symposium, to be published in the 1998 ASME Design For Manufacturing Symposium.

[108] F. N. Woods. *Higher Geometry.* Dover Publications Inc, 1922.

[109] J. R. Woodwark. *Computing Shape.* Butterworths, 1986.

[110] C. S. Zhao, M. Farooq, and M. M. Bayoumi. Analytical solution for configuration space obstacle computation and representation. In *Proceedings of the 1995 IEEE 21st International Conference on Industrial Electronics, Control, and Instrumentation.*, volume 2, pages 1278–1283. IEEE, Los Alamitos, CA, USA, 1995.

[111] D. J. Zhu and J. C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7(1):9–20, 1991.