

DIMENSIONAL INSPECTION PLANNING
FOR COORDINATE MEASURING MACHINES

by

Steven Nadav Spitz

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 1999

Copyright 1999 Steven Nadav Spitz

To Diane, my parents, and my teachers.

Acknowledgments

I am deeply grateful to Professor Aristides Requicha for his guidance and supervision during my studies. His extensive background in programmable automation and keen mind were invaluable sources throughout this work. I would like to thank Ari for always being accessible and for demonstrating a strong work ethic.

I would also like to thank Professor Ari Rappoport for introducing me to geometric modeling and computer graphics. Ari believed in my abilities while I was an undergraduate at Hebrew University, and encouraged me to pursue my studies.

Thanks to all past and present members of the programmable automation laboratory at USC, and the members of the laboratory for molecular robotics next door. Special thanks go to Cenk Gazen for sharing his passion for mathematics, computing, and playing tennis, and Kusum Shori for her invaluable help with administrative duties.

Finally, I would like to thank all my family and friends for their encouragement and support. My wife has been a constant source of love and inspiration. Thanks for carrying me through this, Diane, and making sure that I have my weekends off.

Contents

	ii
Acknowledgments	iii
List Of Tables	vii
List Of Figures	viii
Abstract	x
1 Introduction	1
1.1 Overview	1
1.2 Related Work	3
1.3 Contributions of this Dissertation	7
1.4 Outline	8
2 Accessibility Analysis	9
2.1 Introduction	9
2.2 Related Work	10
2.2.1 Accessibility Analysis	10
2.2.2 Cubic Maps	11
2.3 Tip Accessibility	11
2.4 Straight Probes	12
2.4.1 Half-line Probes	13
2.4.2 Grown Half-lines	15
2.4.3 Ram Accessibility	17
2.4.4 Surface Accessibility	20
2.4.5 Path Accessibility	23
2.5 Bent Probes	23
2.5.1 First Component Accessibility	25
2.5.2 Second Component Accessibility	26
2.5.3 First Component Accessibility Revisited	28
2.6 Summary and Conclusions	29

3	High-Level Planning	33
3.1	Introduction	33
3.2	Problem Statement	34
3.2.1	Input (CMM and Toleranced Part)	34
3.2.2	Measurement Graph and Measurements	34
3.2.3	Output (HLIP-tree)	36
3.3	Planning by Constraint Satisfaction	39
3.3.1	Variables (Measurements)	39
3.3.2	Constraints (Approachability)	39
3.3.3	Existence of Solutions	40
3.3.4	Constructing Plans from CSP Solutions	40
3.3.5	Representing Domains	41
3.4	Plan Quality and <i>Same</i> Constraints	43
3.4.1	A Constraint Hierarchy for Plan Efficiency	45
3.4.2	A Constraint Hierarchy for Plan Accuracy	46
3.4.3	Efficient Plans of High Accuracy	48
3.5	Hierarchical Constraint Satisfaction	49
3.5.1	Clustering Up the Measurement Graph	50
3.5.2	Measurement Sub-Graph Extraction	51
3.5.3	Main Constraint Satisfaction Algorithm	52
3.5.4	First Example	54
3.5.5	Second Example	57
3.5.6	Clustering Algorithm	58
3.6	Closing Loose Ends	60
3.6.1	Preferences for Setups, Probes and Orientations	60
3.6.2	Replacement of Datum Features by Mating Surfaces	63
3.6.3	Segmentation and Point Sampling	66
4	Planner Architecture	69
4.1	Main Algorithm	69
4.1.1	Knowledge Base Initialization	70
4.1.2	Testing Existence of a Solution	71
4.1.3	Plan Validation	71
4.1.4	Incremental Knowledge Acquisition	72
4.2	Speed-Ups	73
4.2.1	Controller	73
4.2.2	Caching Information	75
4.3	Points of Failure	75
4.3.1	Correct Termination	76
4.3.2	Empty D'_1 Cone	76
4.3.3	Fixtures and Other Obstacles	77
4.3.4	Discrete Direction Cones	77
4.3.5	Numerical Precision	79

4.4	Simulation and User Interaction	79
4.5	Implementation and Results	80
4.5.1	Toy Parts	80
4.5.2	Real-World Parts	83
4.5.3	Run-time Results	89
4.6	Conclusions	92
5	Path Planning	93
5.1	Introduction	93
5.2	Related Work	95
5.2.1	General Path Planning	95
5.2.2	CMM Path Planning	97
5.3	General Method	99
5.3.1	Roadmap Construction	100
5.3.2	Optimal Tour Extraction	101
5.3.3	Example	102
5.4	CMM Heuristics	103
5.4.1	The Domain	104
5.4.2	Local Planner	105
5.4.3	Enhancement Step	108
5.5	Implementation and Results	108
5.6	The Localization Problem	111
5.7	Conclusions	113
6	Conclusions	115
6.1	Summary	115
6.2	Contributions	116
6.3	Limitations and Future Work	117
	Reference List	119

List Of Tables

4.1	The input parts	84
4.2	The number of probes and preferred setups	84
4.3	The measurement graphs	84
4.4	Run-time results (fixed head)	90
4.5	Run-time results (orientable head)	90
4.6	Accessibility analysis results (fixed head)	91
4.7	Accessibility analysis results (orientable head)	91
5.1	Path planning run-time results	111

List Of Figures

1.1	A typical coordinate measuring machine	2
1.2	Proposed inspection planner	2
2.1	The offset point	12
2.2	A straight probe and some possible abstractions	12
2.3	The GAC of point p with respect to obstacle X	13
2.4	Experimental results – GAC	14
2.5	Growing a solid	17
2.6	The GAC for the ram component of a straight probe	18
2.7	The viewing volume for truncated half-lines (d, ∞) and $(0, d)$	19
2.8	The variety of GACs for straight probe abstractions	21
2.9	Setup planning with a straight probe	23
2.10	A bent probe and a possible abstraction	24
2.11	Computing D_1 and a portion of D_2	25
2.12	A point that is <i>accessible</i> but not <i>approachable</i> by a bent probe	27
2.13	Computing $D'_1 \subset D_1$ that corresponds to \vec{v}_2	28
2.14	Experimental results – D_1 and D'_1	30
3.1	Example input to the inspection planner	35
3.2	The measurement graph	35
3.3	The simplified measurement graph	36
3.4	A HLIP-tree	37
3.5	A trivial HLIP-tree	40
3.6	Representing the domain of a measurement	41
3.7	The allowable values for the measurements in Figure 3.1	42
3.8	A HLIP-tree representing efficient plans	46
3.9	A HLIP-tree representing plans of high accuracy	47
3.10	A HLIP-tree representing efficient plans of high accuracy	49
3.11	The clustering operation	49
3.12	Clustering up a measurement graph	51
3.13	Sub-graphs and measurement sub-graphs	51
3.14	Measurement sub-graph extraction	51
3.15	Probe selection depends on setup selection	54
3.16	Example domains (CMM has a fixed head)	54

3.17	Clustering the setup orientations	55
3.18	Clustering the probes	55
3.19	The measurement sub-graph for each (\mathbf{s}, \mathbf{p}) configuration	56
3.20	The resulting HLIP-tree	56
3.21	Clustering the setup orientations differently	57
3.22	Clustering (a) setup orientations, and (b) probes	58
3.23	Cluster selection based on a weight function	61
3.24	A single unstable setup vs. two preferred setups	62
3.25	Representing the domain of a datum that is placed on the table	64
3.26	Valid probe orientations for inspecting the table	64
3.27	The measurement sub-graph for a setup that places a datum on the table	65
3.28	Segmentation examples	66
3.29	The measurement graph with fpoints sampled from each face	67
4.1	Domains of neighboring fpoints	74
4.2	Problems with discrete direction cones	77
4.3	Adding preferred directions	78
4.4	A simulation snapshot	80
4.5	HLIP for F2 (fixed head)	82
4.6	HLIP for F3 (fixed head)	82
4.7	HLIP for F3 (orientable head)	82
4.8	HLIP for F4 (orientable head)	82
4.9	HLIP for swiss-block (fixed head)	83
4.10	HLIP for swiss-sphere (orientable head)	83
4.11	HLIP for PolySqrTa (fixed head)	85
4.12	HLIP for PolySqrTa (orientable head)	86
4.13	HLIP for cami2 (fixed head)	87
4.14	HLIP for cami2 (orientable head)	87
4.15	HLIP for nclosurT (fixed head)	88
4.16	HLIP for nclosurT (orientable head)	88
4.17	Inspecting a face that is resting on the table	89
5.1	A multiple-goals path planning problem	94
5.2	A 2-D illustration of the path planning algorithm	103
5.3	The approach/retract path	104
5.4	Plans generated by the local planner	105
5.5	The swept ram and probe	107
5.6	An example part (left) and a section (right)	109
5.7	Path planning with 30 measurement points	110
5.8	Path planning with 100 measurement points	110

Abstract

This thesis documents the development of a fully automated dimensional inspection planner for coordinate measuring machines (CMMs). CMMs are very precise Cartesian robots equipped with tactile probes. Given a solid model of a manufactured part, the goal of dimensional inspection is to determine if the part meets its design specifications. The planner first generates a high-level plan that specifies how to setup the part on the CMM table, which probes to use and how to orient them, and which measurements to perform. This plan is then expanded to include detailed path plans and ultimately a program for driving the CMM.

The planner has been implemented and includes an accessibility analysis module, a high-level planner, a plan validator (through collision detection), a simulator, and a path planner. We tested the planner on real-world mechanical parts and it is sufficiently fast for practical applications.

The accessibility analysis module provides a suite of algorithms to compute global accessibility cones (GACs). GACs are sets of directions along which a probe can contact given points on an object's surface, and are used by the planner to determine part setups and probe orientations. The GAC algorithms make use of widely available computer graphics hardware, and are very efficient and robust.

The high-level planner generates plans by solving a constraint satisfaction problem (CSP), where hierarchical constraints define the requirements of good plans. Plans are extracted using efficient clustering techniques. High-level planning by clustering and without backtracking is a novel approach.

The path planner finds an efficient and collision-free path for the CMM to inspect a set of points. We use a roadmap method to connect the points through simple paths. Then, we find an efficient tour of the roadmap by solving a traveling salesperson problem. The path planner easily integrates CMM heuristics and is probabilistically complete.

Chapter 1

Introduction

1.1 Overview

A Coordinate Measuring Machine (CMM) (Figure 1.1) is essentially a very precise Cartesian robot equipped with a tactile probe, and used as a 3-D digitizer [5]. The probe, under computer control, touches a sequence of points in the surface of a physical object to be measured, and the CMM produces a stream of x, y, z coordinates of the contact points. The coordinate stream is interpreted by algorithms that support applications such as reverse engineering, quality control, and process control. In quality and process control, the goal is to decide if a manufactured object meets its design specifications. This task is called *dimensional inspection*, and amounts to comparing the measurements obtained by a CMM with a solid model of the object. The model defines not only the solid's nominal or ideal geometry, but also the tolerances or acceptable deviations from the ideal [3]. The inspection results are used to accept or reject workpieces (quality control), and also to adjust the parameters of the manufacturing processes (process control).

Currently, computer-controlled CMMs are programmed by tedious off-line simulation or through teaching techniques [51]. This dissertation focuses on *automatic planning and programming* of dimensional inspection tasks with CMMs. Given a solid model of an object, including tolerances, and a specification of the task (typically as a set of features to be inspected), the goal is to generate a high-level plan for the task, and then to expand this plan into a complete program for driving the CMM and inspecting the object (see Figure 1.2). The high-level plan specifies how to setup the part on the CMM table, which probes to use and how to orient them,

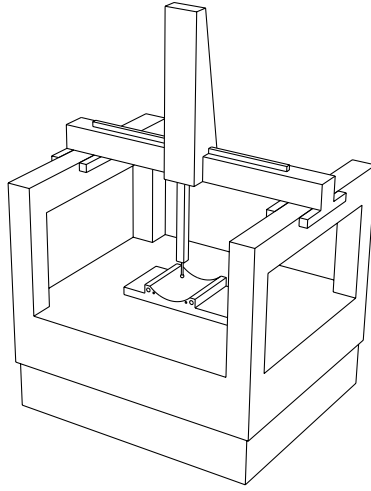


Figure 1.1: A typical coordinate measuring machine

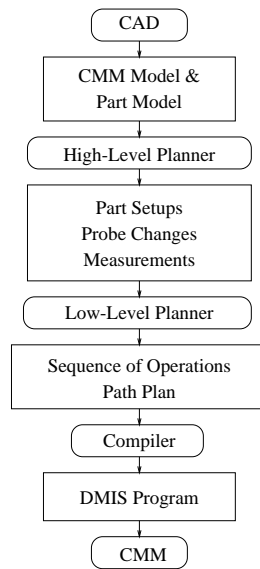


Figure 1.2: Proposed inspection planner

and which surface features to measure with each setup, probe and probe orientation. The final program contains specific probe paths and points to be contacted by the probe tip, and is interpretable by the CMM controller through DMIS code [12, 77]. (DMIS is a national standard for CMM control.)

There are many challenges in building a fully automated inspection planner. First, one must reason about the geometry of the product to determine which operations can be performed. Then this information must be combined to form *good* inspection plans. In this dissertation we describe the components of a planner and show how they are combined into a system capable of dealing with real-world mechanical parts.

1.2 Related Work

Planning is a classic problem in artificial intelligence (AI) [68]. The focus of the AI community has been largely on STRIPS-like domains or extensions of them [1]. These problems have discrete domains and are typically combinatorially hard, i.e., feasible plans are difficult to find. On the other hand, in manufacturing operations planning all the possible operations are precomputed and the problem is to find a *good* plan as opposed to *any* plan [58]. Basically, the problem is partitioned into two parts: (1) geometric reasoning to extract all possible operations, and (2) an optimization problem to generate a good plan.

Plan merging [20] and planning by rewriting [2] are techniques that tackle a generalization of the machining optimization problem. Both methods assume that an initial low-quality plan can be generated cheaply. For example, every problem in the blocks-world domain [1] can be solved trivially in two steps: first unstack all the block onto the table and then build the goal towers from the bottom up. Once an initial plan is generated, then graph rewriting techniques are used to (locally) optimize the plan. The fact that the blocks-world problem can be solved trivially is not a contradiction to the fact that the original problem is combinatorially hard. In the original problem we did not have the meta-knowledge that is needed to generate these plans.

We solve the inspection planning problem using a two step procedure that is similar to manufacturing operations planning [58] — we compute all possible operations

and then construct a good plan through clustering techniques. One main difference is that the machining domain becomes discrete at the process plan level (i.e., during the second, or optimization step), whereas the inspection planning problem remains continuous. Constructing good plans through clustering techniques can be viewed as plan merging in continuous space.

Other research groups have designed inspection planning systems, which will be reviewed in the remainder of this section. We will see that each system concentrates on solving a different aspect of the problem, but none has produced a fully automated planner. For different perspectives on the literature, the reader is referred to two survey papers [16, 39].

One of the earliest publications was written by Hopp and Lau at the National Bureau of Standards [26, 27]. It is not clear from the papers if the system was ever implemented. Nevertheless they made an important contribution by carefully stating the problem and pointing out that AI techniques may be a promising approach to its solution. They defined a hierarchical planning scheme that has been used in much of the following work. The hierarchy is composed of tolerances, features, surfaces, probing points, paths, machine motions and servo commands. The scope of inspection (i.e., which tolerances to inspect) and part setups are determined in the tolerances level. Measurable surfaces are determined at the surface level, and help in sensor planning. After nominal points are selected on the measurable surfaces, a probe path is generated using collision avoidance algorithms. Finally, the plans are translated into machine motions and then into servo commands.

The earliest implemented CMM planner was developed by ElMaraghy and Gu at McMaster University [17]. Their system was a rule-based expert system for turned parts, which are essentially 2-D. They were able to develop rules for simple accessibility analysis. For example, a straight probe cannot access a hole that is below another one of smaller diameter. They also developed rules that filter out features that should be inspected with non-CMM devices. The algorithm does not deal with optimization issues and follows a simple logic: select a part setup, select a probe and find all accessible features, repeat the process for the next available probe until all features are inspected, otherwise change the setup and repeat for the features that were not inspected yet. It is clear that the plan generated with this method can be far from optimal. Furthermore the search may take a long time if the setups are

not selected intelligently. The papers do not specify how setups are selected. The system also suffers from a problem that is common to most expert systems: there are always exceptions to the available rules, causing the system to fail or the set of rules to become large and unmanageable.

Brown and Gyrogo developed IPPEX [7, 8], which is an enhancement of the EPS-1 project [62]. The system is knowledge-based and uses a hierarchical planning strategy similar to that described by Hopp and Lau [26, 27]. Most of the effort in this project involved the development of the system architecture and a convenient user interface that allows manual intervention. Many of the specified modules were not implemented, thus inhibiting full automation.

APM is an inspection planning system developed by work funded by the Department of Energy [34]. It is a partially automatic system that uses rule-based methods to alleviate the burden involved in the planning. The rules were developed from a series of interviews with inspection experts and are limited to a small number of tolerance types. The published description does not go into implementation details. The authors conclude that “of great concern is the possibility that a sufficient robust rule base cannot be built to handle the large variety of (inspection planning) tasks...”. This statement applies equally well to the expert systems described previously. Nevertheless rule-based expert systems can play an important role in specific inspection planning tasks, such as eliminating non-CMM operations as in the McMaster system [17].

Another inspection planning system was developed by Merat and Radack at Case Western Reserve University as part of a *rapid design system* [54, 55]. The planner uses a feature-based CAD model of the part, and exploits design information in a bottom-up fashion. Every feature is issued an inspection plan fragment (IPF) that is used to find a list of suitable inspection points. These fragments are combined to generate an inspection path for the CMM. The power of this approach stems from the fact that IPFs are determined locally. Each feature has a macro that generates the appropriate IPF based on feature parameters. The weakness, however, is that global interaction between features may invalidate otherwise suitable IPFs. The authors attempt to deal with interactions by generating IPFs surface by surface. However, their papers do not describe how they detect interacting features and how they generate inspection plans for them.

Spyridi and Requicha describe an inspection planning system developed at the University of Southern California [75, 76, 72]. They use a two-level hierarchical planning strategy. The high-level planner determines the part setups, the probes and the probe orientations for the surface features to be inspected. The low-level planner refines the plan by selecting measurement points and creating the CMM path plan. Spyridi and Requicha focus on high-level planning and use a least-commitment planning strategy and a problem-space architecture to search the space of incomplete inspection plans. They take a novel approach in defining incomplete inspection plans as special trees that enforce constraints, called *same* constraints, to control the efficiency and accuracy of the resulting plan. Operators on these trees refine the inspection plan by performing computations at different levels of cost. These operators transform one tree to another and are used to search plan space. Accessibility analysis plays a crucial role in the system. Global accessibility cones, which consist of the set of directions from which an abstract straight probe can inspect a surface feature, are computed by geometric algorithms. Part setups are computed by clustering these direction cones to find a minimum number of setups that suffice to inspect the part. Features may be replaced in certain cases by the CMM table or fixture surfaces. The system has several limitations. First, it is not clear how *same* constraints can be applied automatically - the problem can quickly become over constrained. Second, the system relies on Minkowski operations for accessibility analysis. These require polyhedral approximations and tend to be slow. Third, the implemented system's control structure involved human intervention, and could only be tested with very simple parts. The work reported in this thesis is based on Spyridi and Requicha's planner, but extends it non-trivially and uses much more efficient algorithms.

Gu and Chan developed an inspection planner at the University of Saskatchewan, Saskatoon [23]. Their system is divided into a high-level inspection process planner (IPRP) and a low-level inspection path planner (IPAP). The IPRP uses a fixed hierarchy of tasks: initial plan generation of possible part setups, accessibility analysis of features by specific probes in specific setups, selection of part setup and probes, and sequencing of features to be inspected. Their system seems to be limited to straight probes and a small number of possible setups - the 6 major axes of the

part's coordinate system. These restrictions enable the system to search over the range of setups and probes.

Other systems have been developed that concentrate primarily on the low-level path planning problem. Probably the most noticeable is the hierarchical planner at Ohio State University [41, 52, 87]. The published descriptions give very little detail on how setups and probes are selected. They concentrate on the details of path planning for inspection of free-form surfaces. Work related to path planning is reviewed in Chapter 5.

It should be noted that computer vision inspection systems have been developing in parallel to CMMs [61, 50, 85, 79]. Accessibility analysis for straight probes is very similar to the visibility problem [59], which is crucial for visual inspection. Work related to accessibility is covered in Chapter 2.

1.3 Contributions of this Dissertation

In this work we develop a completely automated inspection planner. The number of assumptions regarding the toleranced product are kept to a minimum. For example, the planner can handle parts with free-form surfaces. The only assumption is that the part surface can be approximated by a mesh, which is a standard facility in most modern geometric modelers [70]. We make no assumptions on the size of the product or the corresponding CMM. Currently, our implementation is limited to straight and bent probes (i.e., CMMs with fixed head or orientable heads), but the general architecture can handle other probes, e.g., star probes.

To the best of our knowledge, this is the first fully automated inspection planner. The planner generates the part setups, selects the probes to be used, decides on the probes' orientations, selects measurement points and computes a path plan. Other planners make simplifying assumptions, such as a predefined set of setups. Our planner allows the input of preferred setups, which can be used to guide the planner, but are not necessary for the planner to work. These preferred setups can be generated automatically through external agents, such as a stability analysis module.

We provide a simulator that is used to visualize the inspection plan and edit it if necessary before re-planning. We are aware of the limitations of a completely

automatic system and allow editing tools, so that the operator can guide the planner if the automatically computed plans are judged unsatisfactory. The planner comes with a plan validator (collision detector) to ensure that the generated plans are correct.

We make a clear formulation of high-level inspection plans as solutions to a constraint satisfaction problem (CSP), and describe a clustering method for extracting plans of high quality. A plan is valid if all the measurement points are *approachable*. We approximate approachability through accessibility analysis and provide efficient algorithms to compute accessibility for CMM probes. (Approachability and accessibility are defined formally in Chapter 2.) The algorithms are simple and efficient, and exploit standard computer graphics hardware.

Finally, we provide a general framework for solving the path planning problem. The idea is to build a roadmap of feasible paths between every pair of measurement points, and then extract a short path by solving the traveling salesman problem on this roadmap. The path planner exploits the typically dense distribution of measurement points, which implies that it is easy to construct a connected roadmap.

1.4 Outline

In Chapter 2 we introduce accessibility analysis as a tool for computing tentative setup and probe orientations. We provide practical algorithms that make use of standard computer graphics hardware. Chapter 3 formulates the high-level inspection planning problem as a constraint satisfaction problem. This chapter concludes with an algorithm that generates high-level inspection plans of good quality assuming that the domains of the variables in the CSP are known. However, this knowledge is typically incomplete, because it involves expensive geometric computations. Therefore, Chapter 4 describes the planner architecture used to generate plans with incomplete knowledge. A prototype implementation and experimental result are described. Finally, Chapter 5 discusses the path planning module, which refines a high-level plan into a low-level plan. We conclude the dissertation with a discussion of contributions and future work.

Chapter 2

Accessibility Analysis

2.1 Introduction

Reasoning about space is crucial for planning and programming of tasks executed by robots and other computer-controlled machinery. Accessibility analysis is a spatial reasoning activity that seeks to determine the directions along which a tool or probe can contact a given portion of a solid object's surface. For concreteness, this chapter discusses accessibility in the context of automatic inspection with Coordinate Measuring Machines (CMMs), but the concepts and algorithms are applicable to many other problems such as tool planning for assembly [82, 83], sensor placement for vision [50, 79], numerically controlled machining [10, 25], and so on.

Exact and complete algorithms for accessibility analysis in the domain of curved objects are either unknown or impractically slow. We present in this chapter several accessibility algorithms that make a variety of approximations and trade speed of execution for accuracy or correctness. Some of the approximations are pessimistic, i.e., they may miss correct solutions, typically as a result of discretizations. Other approximations are optimistic and may sometimes produce incorrect solutions. (These will eventually be rejected when the plan is tested.) The algorithms described here have been implemented and tested on real-world mechanical parts, and have been incorporated in our inspection planner (see following chapters).

The remainder of the chapter is organized as follows. First, related work is briefly reviewed. Next, we discuss accessibility for the tips of probes. Then accessibility for the case in which probes are straight, i.e., aligned with the CMM's ram. Then we

consider bent probes, which consist of two non-aligned components. A final section summarizes the chapter and draws conclusions.

2.2 Related Work

2.2.1 Accessibility Analysis

Spyridi and Requicha introduced the notion of accessibility analysis as a tool for high-level inspection planning for CMMs [72, 73, 74]. Their implementation computed *exact* global accessibility cones (GACs, defined below) for planar faces of polyhedral parts using Minkowski operations. Sets of directions, called direction cones, were represented as 2-D boundaries on the unit sphere and GACs were computed by projecting elements of the Minkowski sum onto the sphere. Their algorithm proved to be impractical for complex parts with curved surfaces.

Other researchers computed GACs at single points, thus eliminating the need of computing Minkowski sums. This is the approach that we take as well. Lim and Menq used a ray casting technique with an emphasis on parts with free-form surfaces [41]. Limaïem and ElMaraghy developed a method to compute GACs that used standard operations on solids [42, 43]. A similar technique was independently developed by Jackman and Park [28]. Medeiros *et al.* use visibility maps, which provide a representation for non-homogeneous direction cones [36, 89, 90]. All of the above methods are too slow for practical inspection planning, where many accessibility cones must be computed for complex objects.

Accessibility analysis is related to work in other fields. The visibility problem is a generalization of the global accessibility problem, because directions of accessibility correspond to points of visibility at infinity [59, 60, 15]. Sensor placement in visual inspection systems is related to the problem of straight probe accessibility [50, 78, 79]. Other fields which require accessibility analysis for high-level task planning include assembly planning [82, 83] and numerically-controlled machining [10, 25, 30, 80, 84].

We are not aware of previous work involving accessibility analysis of *bent* probes as introduced in Section 2.5. The theoretical foundations for bent probe accessibility appear in [72], and a rigorous mathematical analysis of accessibility in [71].

2.2.2 Cubic Maps

We use a cubic mapping of the unit sphere to represent direction cones, i.e., subsets of the unit sphere. This technique has been used in other areas of computer graphics, such as radiosity, shadow computations and reflections. This is not surprising, because global accessibility is strongly related to global visibility, as noted above.

Environment (or reflection) maps ([19], pg. 758-759) are a generalization of the direction-cone map presented here. The environment map holds color images, while the direction-cone map holds bitmaps. The light-buffer ([19], pg. 783) is another cubic map that is used to partition the space visible to a light source. The hemi-cube structure ([19], pg. 795-799) is used to determine visibility between surface patches and calculate their contribution to the radiosity equation. Unlike our direction-cone mapping, the hemi-cube is not aligned with the world coordinate system. It is aligned with each patch, and therefore is not suitable for Boolean operations between direction cones.

Shadow maps ([19], pg. 752) are depth images of a scene as viewed from a light source. These are used to compute the space that is visible to a light source in order to apply global shading. This is not a cubic map, but the technique used in the two-pass z-buffer shading algorithm ([19], pg. 752) is similar to our method of extracting the first-component directions of a bent probe (Section 2.5.3).

2.3 Tip Accessibility

A CMM has a touch-trigger (or tactile) probe with a spherical tip. We define the origin of the probe to be the center of the tip. The CMM measures the spatial coordinates of the tip's center when the tip comes in contact with an obstacle. We say that a point p is *accessible* to a tip with respect to an obstacle X if the tip does not penetrate X when its origin is placed at p . With CMMs, the obstacle X is normally the workpiece to be inspected. (Fixturing devices and other obstacles are ignored here, because they are not relevant to accessibility analysis in the early stages of inspection planning.)

Testing tip accessibility is a simple matter of placing the tip at p and checking for collisions with the obstacle. Notice that if p is the point to be measured on the

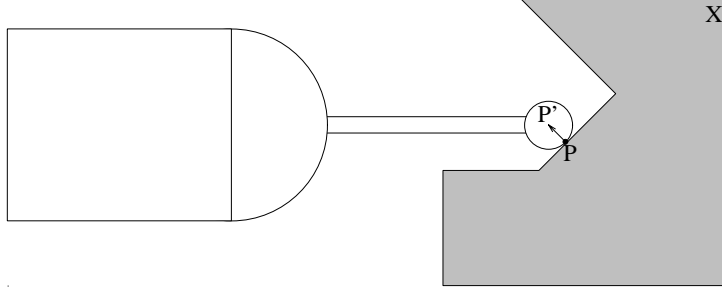


Figure 2.1: The offset point

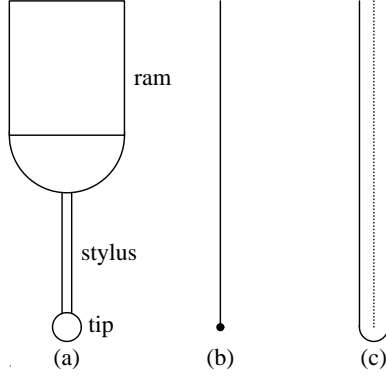


Figure 2.2: A straight probe and some possible abstractions

surface of the workpiece, then placing the center of the tip at p will cause the probe to penetrate the part. Instead, we perform accessibility analysis for the offset point $p' = p + r\vec{n}$ (see Figure 2.1), where r is the radius of the tip and \vec{n} is the normal to the surface at p . (We assume that p is not singular, because it is not wise to measure a singular point with a tactile probe.) For the remaining of this chapter, we ignore these issues and assume that p is the offset point to be accessed by the center of the tip.

We assume that the CMM has a small number of probes, therefore testing the accessibility of each tip at each point is reasonable. See [56] for an alternative approach.

2.4 Straight Probes

A *straight probe* (Figure 2.2a) is attached to the CMM ram (Figure 1.1), which is much longer than the probe and aligned with its axis. In the remainder of this

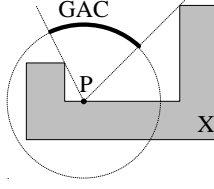


Figure 2.3: The GAC of point p with respect to obstacle X

dissertation we refer to the whole ram/probe assembly as a straight probe and assume in this case that the CMM has a *fixed* head, such as the Renishaw PH6 [64]. In general, the straight probe can be any tool, not necessarily a CMM probe, that can be considered symmetric about an axis for the purpose of accessibility analysis. Examples of such tools are drills, screw drivers and laser range finders.

On the axis of the tool we define a point that is the origin of the tool. *Accessibility analysis for a point p with respect to an obstacle X seeks to determine the directions of the tool axis such that the tool does not penetrate X when the tool's origin is placed at p .*

In this section we investigate the accessibility of a point by several straight probe abstractions. Then we generalize to the accessibility of surfaces and briefly outline how to apply the results to setup planning for dimensional inspection with CMMs.

2.4.1 Half-line Probes

Consider a straight probe abstracted by a half-line that is the main axis of the probe (see Figure 2.2b). This is an optimistic abstraction of the probe, because it ignores the fact that the probe has volume, but it captures the fact that the CMM ram is typically very long. Furthermore, it is simplistic enough to give rise to efficient algorithms.

We say that a point p in the presence of obstacle X is *accessible* if the endpoint of a half-line can be placed at p while not penetrating X . The direction of such a half-line is called an *accessible direction*. The set of all accessible directions is called the *global accessibility cone* (GAC) of point p with respect to obstacle X , and is denoted by $GAC(X, \{p\})$.

Figure 2.3 illustrates the global accessibility cone of a point p with respect to an obstacle X . $GAC(X, \{p\})$ is the highlighted portion of the unit sphere centered at p .

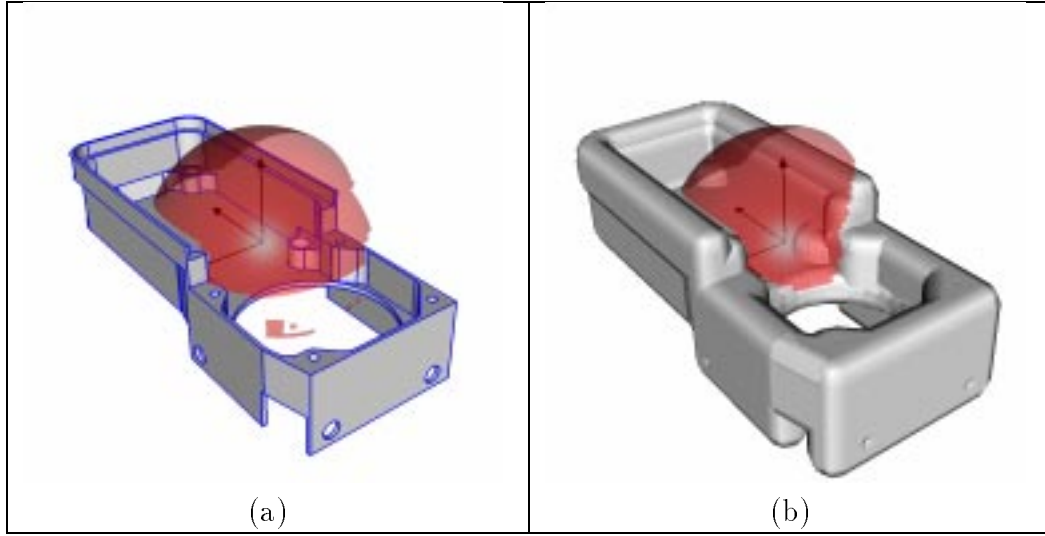


Figure 2.4: Experimental results – GAC

It is easy to verify that the GAC complement is the projection of X onto the sphere. This forms the basis for our algorithm to compute the GAC of a point: project the obstacle onto a sphere centered at p and take the complement.

The global accessibility cone is computed in the same fashion as environment maps [19]. The obstacle (i.e., environment) is projected onto the faces of a cube centered at p . The cube is aligned with the world coordinate system, and each face is a bitmap. The algorithm first sets all the bits of the cube to 1, then renders the obstacle as 0s in order to delete the projected obstacle from the direction cone. The remaining directions are the desired complement set that form the GAC.

The cubic map is used as a non-uniform partition of the unit sphere. There is a one-to-one mapping between directions and unit vectors in 3-D space, therefore *the cubic map is a discrete representation for a set of directions or a direction cone*. The mapping between a direction and a bit on the cube involves selecting the appropriate face of the cube, projecting the unit vector (i.e., direction) onto the face, and normalizing the result to bitmap coordinates. The direction falls on the face of the cube that lies on the axis of the largest coordinate of the (x, y, z) unit vector, with the sign of this coordinate used to distinguish between the two opposing faces.

Figure 2.4a shows the result of running our algorithm on a real-world mechanical part, which was modeled in ACIS [70]. The GAC is projected onto a sphere that is centered about the point of interest (in the center of the figure). As a preprocessing step, the ACIS faceter produced the mesh that was used to render the mechanical part. This mesh is a collection of convex polygons that were not optimized for rendering other than being placed in an OpenGL display list. The mechanical part contains 103 faces (including curved surfaces) and the mesh contains 1980 polygons. The code executed on a Sun ULTRA 1 with Creator 3D graphics hardware, Solaris 2.1 and 128 MB of memory. Direction cones were represented by six 32×32 bitmaps, for a total of 6144 directions at a cost of 768 bytes. The running time for the algorithm was 0.08 seconds. Not surprisingly, most of the load was on the graphics hardware.

The complexity of the algorithm described above depends solely on the time to render the obstacles. The obstacle X may be rendered many times to compute GACs at different points, so it is wise to optimize the mesh used to display X . For example, one can use triangle strips [69].

2.4.2 Grown Half-lines

In the previous section we abstracted a straight probe by a half-line. Here we generalize this to a half-line that is grown by a radius r (see Figure 2.2c). An object grown by a radius r includes all the points that are at a distance no greater than r from another point in the object [67]. It also equals the Minkowski sum of the object and a ball of radius r . Thus, a grown half-line is a semi-infinite cylinder with a hemi-sphere over the base. This leads to a straight probe abstraction that can serve as an envelope for the *volume* of a probe, and therefore is a pessimistic approximation.

It is easy to verify that a half-line grown by a radius r penetrates an obstacle X iff the non-grown half-line penetrates $X \uparrow r$, where $X \uparrow r$ denotes X grown by r . (This is a well known result in robot motion planning [37].) In other words, $GAC(X \uparrow r, \{p\})$ describes the directions from which a point p is accessible by a half-line grown by r .

A straightforward algorithm to compute $GAC(X \uparrow r, \{p\})$ is to compute the grown obstacle X and to apply the algorithm presented previously in Section 2.4.1. Unfortunately, computing the solid model of a grown object is an expensive and non-trivial task that is prone to precision errors [67] and produces curved objects even when the input is polyhedral. If the accessibility algorithm is to be applied many times and for a small set of given radii, then it may be wise to compute the grown solids as a preprocessing step.

We choose an alternative approach in which we *implicitly* compute the grown obstacle, $X \uparrow r$, as it is rendered. The main observation is that only the silhouette of the obstacle is needed as it is projected onto the cubic map. Therefore, we render a *superset* of the boundary of the grown object that is also a *subset* of the grown object itself. The naive approach is to render each vertex of the mesh as a ball of radius r , each edge as a cylinder of radius r , and to offset each polygon by a distance r along the normal. This algorithm is correct and can be optimized by not rendering concave edges, which will not be part of the grown obstacle’s boundary. A more drastic optimization can be applied, if the mesh is partitioned into face sets, each corresponding to a face of the original solid model. Then a facial mesh is represented by an array of nodes and an array of polygons. Each node corresponds to a point on the face and the normal *to the face* at this point. Each polygon is represented as a list of nodes. To offset a facial mesh, first we offset the nodes by translating each point along its normal, and then we render the polygons at these offset nodes. The gain is that the edges and vertices that are internal to a facial mesh do not need to be rendered as grown entities. The only vertices and edges of the mesh that are rendered are those which fall on actual edges of the solid model (see Figure 2.5).

The spheres and cylinders are rendered as quad strips [69] of very low resolution to maximize rendering speed. The cylinders contain 6 faces with no tops or bottoms, because they are “capped” by spheres on either end. The spheres are composed of 3 stacks of 6 slices each (latitude and longitude, respectively). Our results show that these approximations are adequate for practical problems. Finer approximations can be used for more accurate results. The running time to compute the GAC of Figure 2.4b is 1.3 seconds.

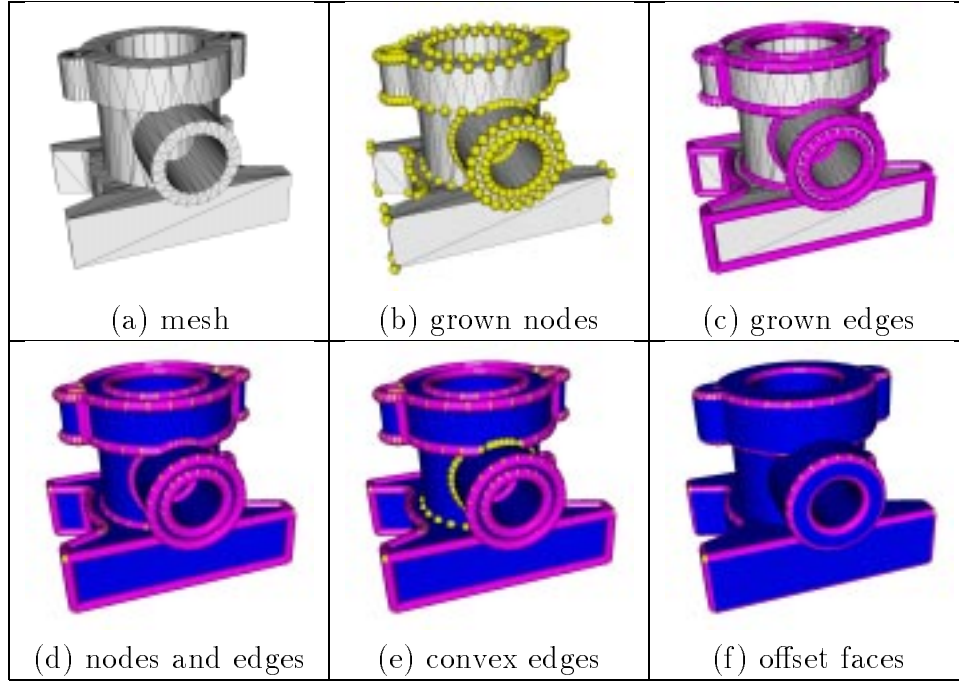


Figure 2.5: Growing a solid

2.4.3 Ram Accessibility

The straight probe abstractions introduced so far have constant thickness. In practice, the CMM ram is considerably fatter than the probe stylus. An improved probe abstraction will take this fact into account as shown in Figure 2.2a. In this case, the probe is modeled by two components, two dilated half-lines that are aligned with each other, one to model the ram and the other to model the stylus. Notice that in order for such a probe to access a point both the ram and the stylus must be able to access it. In other words, the GAC of such an abstraction is the intersection of the GACs of each component of the probe. In this section we focus on the ram component.

We model the ram by a *truncated* half-line (d, ∞) that is grown by a radius r (Figure 2.6a). A truncated half-line (d, ∞) includes all the points on the half-line at a distance no less than d from the origin. Using similar arguments as in the previous sections, the GAC for a ram with respect to an obstacle X is identical to the GAC of the ram shrunk by r with respect to $X \uparrow r$. The shrunk ram is precisely the

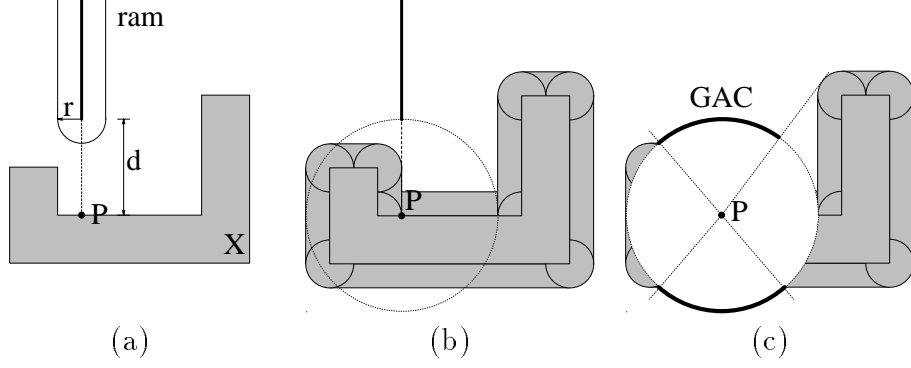


Figure 2.6: The GAC for the ram component of a straight probe

truncated half-line (d, ∞) (Figure 2.6b). We already know how to render $X \uparrow r$, therefore we reduced the problem to computing the GAC for a truncated half-line.

It is clear from Figure 2.6b that the GAC of a truncated half-line (d, ∞) is the GAC of a half-line but with a different obstacle. The idea is to remove the irrelevant region from the obstacle. A truncated half-line (d, ∞) positioned at p cannot collide with any portion of the obstacle that is at a distance closer than d from p . In other words, the ball of radius d that is centered at p can be removed from the obstacle. The GAC with respect to the new obstacle corresponds to the GAC of the truncated half-line (Figure 2.6c).

Calculating the GAC of a truncated half-line (d, ∞) then entails subtracting a ball centered at p from the obstacle and using our algorithm for regular GACs. However, computing the solid difference between an obstacle and a ball is an expensive computation that we wish to avoid. In addition, we do not have a solid model of the grown obstacle itself (see previous section). Consequently, we choose an alternative approach in which we use clipping operations to approximate the solid difference. The clipping is performed during the projection of the obstacle within the GAC algorithm, by introducing a read-only depth-buffer that is initialized with a spherical surface of radius d . This is the portion of the sphere that is visible through each face of the cubic mapping (the depth values are symmetrical for each face). If the depth-buffer is enabled with a “greater-than” comparison, then the clipping operation will approximate the subtraction of a ball of radius d , as needed.

Notice that, in general, the clipping operation is an operation between surfaces and not a Boolean operation between solids [66]. In our case, we position the far

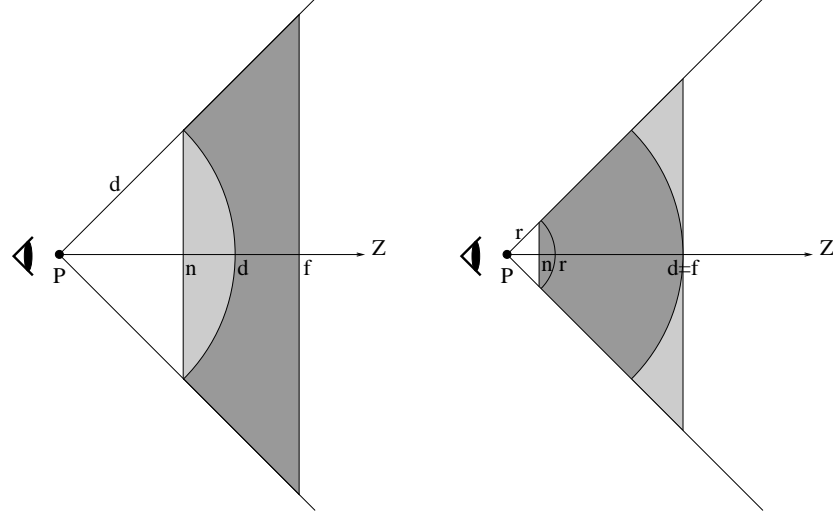


Figure 2.7: The viewing volume for truncated half-lines (d, ∞) and $(0, d)$

clipping plane beyond the obstacle. This ensures that the projection of the solid difference is correct, because a truncated half-line (d, ∞) positioned at p intersects X iff it intersects the boundary of X . Therefore, the point of intersection is rendered along with the boundary of X and it is not clipped, because it is within the viewing frustum and not closer than d to the viewer.

The quality of the approximation depends on the depth-buffer precision. To maximize the precision of the depth-buffer, the distance between the far clipping plane and the near clipping plane should be minimized. Therefore, the far clipping plane should be a tight bound on the obstacle — we use the diameter of the bounding box as a reasonable bound. The near clipping plane is set to a distance of $d/\sqrt{3}$, so that the near face of the viewing frustum is contained in the ball of radius d .

Figure 2.7 shows the viewing volume through a single face of a cube centered at p . The size of the cube is irrelevant, since the result is *projected* onto the face. The near and far clipping planes are $z = n$ and $z = f$, respectively. The viewing volume is bound by the clipping planes and is shaded in the figure. The lighter shade corresponds to the volume clipped by the depth-buffer, which is initialized to be the sphere of radius d centered at p . The left hand side of the figure is the desired viewing volume for a truncated half-line (d, ∞) . The far clipping plane is assumed to be beyond the obstacle. It is easy to see that the near clipping plane must satisfy $n = d/\sqrt{3}$.

The complexity of the algorithm is identical to the regular GAC algorithm with the addition of depth-buffer comparisons and the overhead of initializing the depth buffer with a spherical surface. The depth-buffer comparisons are performed in hardware and should have negligible run-time overhead. Notice that the depth-buffer is read-only, thus it needs to be initialized only once. In addition, the same depth buffer may be used for all probes that have a stylus of length d . Therefore, the cost of initializing the depth-buffer may be amortized over many direction cones.

Our results show that the cost of computing the GAC for a truncated half-line is identical to the cost of computing a regular GAC. The cost of initializing the depth-buffer is negligible, because the buffer is relatively small (32×32 bits).

To review, Figure 2.8 illustrates the GACs computed with our system for a simple “L” shaped obstacle. The left column shows the GACs for the original obstacle, and the right column shows the GACs with respect to the obstacle grown by a constant radius. Figure 2.8(c’) shows the GAC for a truncated half-line (d, ∞) with respect to a grown obstacle, which is precisely the GAC for a ram. The last row of this figure illustrates the GACs for truncated half-lines $(0, d)$, which will be introduced in Section 2.5.1.

To aid visualization, the 3-D cones have been rendered with transparent material. For example, the GAC in Figure 2.8b has 3 shades of gray. The lightest shade of gray is on the bottom of the cone. This portion includes the directions that go out of the page and downward. The top portion of the cone is darker, because it includes both outward directions and inward directions. In other words, two surfaces overlap. They are both rendered, because the cone is transparent. The intermediate shade of gray (in the nearly rectangular region) only includes directions that go into the page and away from the protrusion on the obstacle.

2.4.4 Surface Accessibility

Up to this point we have discussed the accessibility of a single point. Now, we extend the notion of accessibility to arbitrary regions of the workspace, which we call *features*. For dimensional inspection, these are normally surface features on the boundary of a workpiece. The goal is to find the set of directions from which a straight probe can access *all* the points of a feature.

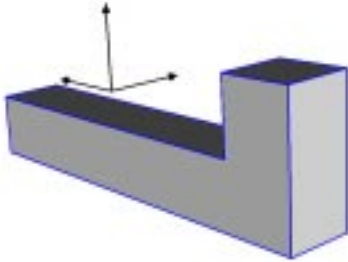
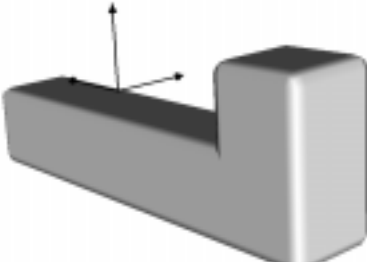
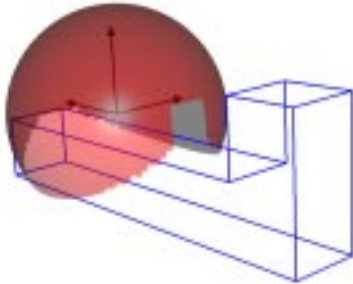
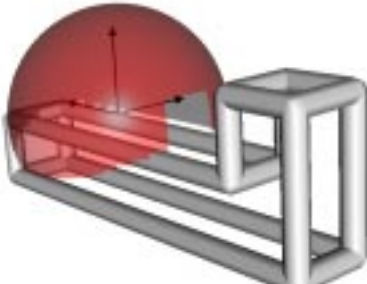
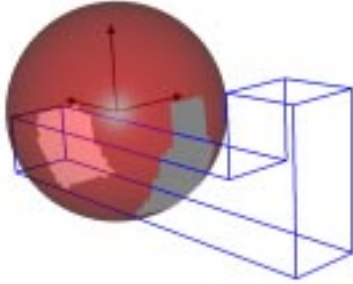
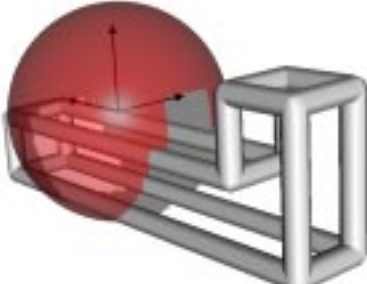
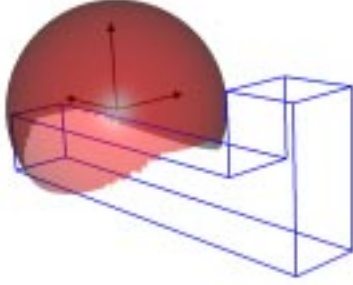
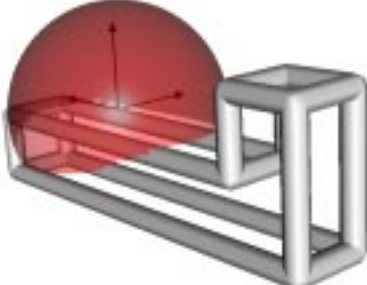
	regular obstacle	grown obstacle
	 (a)	 (a')
half-line	 (b)	 (b')
truncated (d, ∞)	 (c)	 (c')
truncated $(0, d)$	 (d)	 (d')

Figure 2.8: The variety of GACs for straight probe abstractions

The global accessibility cone of a feature F with respect to an obstacle X is denoted by $GAC(X, F)$, and corresponds to the directions from which *all* the points in F can be accessed by a half-line. Clearly, this cone is the *intersection* of the GACs for all the points in F . Notice that $GAC(X, \{p\})$ is a special case — the GAC of a feature containing a single point p .

Exact GACs for planar surfaces and polyhedral obstacles can be computed using Minkowski operations [71]. Algorithms for Minkowski operations are expensive, do not scale well, and are not available for curved surfaces. We choose an alternative approach in which we sample a few points from F and compute the intersection of the GACs at these points. This approximation is especially suitable for CMMs, which are normally restricted to inspect discrete points. In addition, computing the intersection of direction cones represented by cubic maps is an efficient and trivial operation on bitmaps. The approximation is optimistic because it is a lower bound on the intersection of the infinite number of GACs for all the points of the feature.

Notice that the direction of the straight probe corresponds to the direction of the CMM ram with respect to the workpiece. Therefore, we can use this direction to represent the orientation of the workpiece *setup* on the CMM table. Computing the GAC for a feature translates to finding the set of setup orientations from which the entire feature can be inspected. If the GAC of a feature is empty, then this feature cannot be inspected in a single setup orientation. (In this case the feature is segmented into sub-features or a different probe is used.)

For dimensional inspection planning, computing the GACs for all surface features that need to be inspected may be the first step of a high-level planner. Clustering these GACs can produce a minimum number of workpiece setup orientations. This is a very important characteristic, since each setup change is usually a time-consuming manual operation. There are other considerations for setup planning, which will be covered in Chapter 3.

Figure 2.9 shows the usefulness of accessibility analysis for spatial reasoning. In this example, we computed the GACs of all the faces of the workpiece. The cones were partitioned into three clusters, each cluster composed of cones whose intersection is not empty. A direction was chosen from each cluster. The result is that each face can be accessed from at least one of these directions. The directions

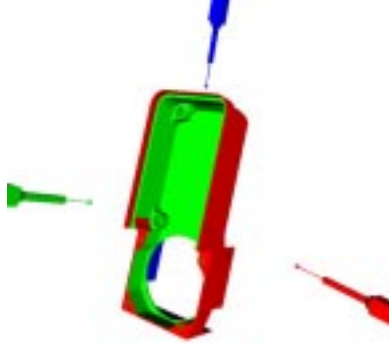


Figure 2.9: Setup planning with a straight probe

of the probes and the faces on the part are color coordinated (or in different shades of gray) to illustrate which faces a probe can access.

2.4.5 Path Accessibility

When the CMM inspects a point, the probe normally traverses a short path, which we call an *approach/retract* path. This path is along a line segment that is normal to the point of contact and is proportional in length to the size of the tip. The probe will approach the point in a slow motion along this path and then retract. The idea is to minimize crashes at high speed and to maximize the accuracy of the measurement.

The goal is to find the set of directions from which a probe can access *all* the points along the approach/retract path, so that the entire path (minus the endpoint) is collision free. Notice that the path can be viewed as a feature in the system, thus all the arguments from the previous section hold true. Since the approach/retract path is typically short relative to the size of the workpiece, it is reasonable to approximate this path by its two end points. Again, this is an optimistic approximation.

2.5 Bent Probes

Orientable probes, such as the Renishaw PH9, are more expensive than straight probes, but much more versatile, and are used often in CMM inspection. The probe can be oriented in digital steps under computer control. We consider the whole ram/probe assembly as forming a *bent probe*, which is not necessarily aligned with

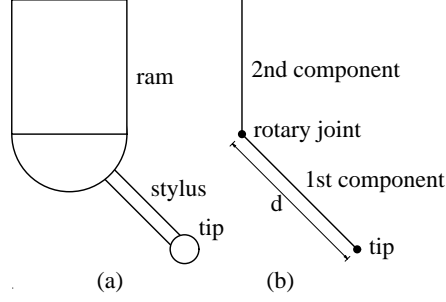


Figure 2.10: A bent probe and a possible abstraction

the ram. A bent probe is a linked chain of two components that are connected at a 2 degrees-of-freedom rotary joint.

We model the probe by a 2-component abstraction as in Figure 2.10b. The first component is a truncated half-line $(0, d)$ straight probe abstraction, which includes all the points of a half-line at a distance no greater than d from the origin. The center of the tip of the bent probe coincides with the tip of the first component. The second component is a half-line or straight probe abstraction with endpoint at a distance d along the axis of the first component. This endpoint corresponds to the probe's rotary joint.

For the rest of this section we assume that a bent probe has no volume, i.e., it is modeled by (truncated) half-lines. Similar generalizations, such as grown half-lines, can be introduced very easily in the manner of Section 2.4.2. Additionally, one can generalize the bent probe concept to more than 2 components, but we will not do so here.

The length of the first component, d , is a constant. Therefore, we can describe the configuration of a bent probe by using a pair of directions — one for each component of the probe. The result is that a GAC for a bent probe is a 4-D cone. Fortunately, applications normally need the second component directions rather than the entire 4-D cone. For example, as with straight probes, the directions of the second component are used to find a minimal number of orientations for setting up the workpiece on the machine table.

The remainder of this section is outlined as follows: Section 2.5.1 introduces the GAC for the first component of a bent probe, Section 2.5.2 shows how to compute

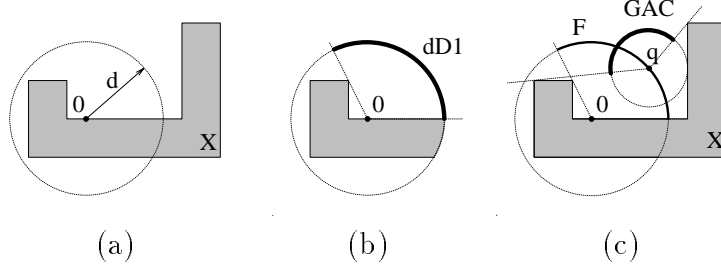


Figure 2.11: Computing D_1 and a portion of D_2

the GAC for the second component of a bent probe, and Section 2.5.3 shows how to compute the first component accessibility given a direction of the second component.

2.5.1 First Component Accessibility

The first component of a bent probe is a truncated half-line $(0, d)$. This is the complement of the ram abstraction that was introduced in Section 2.4.3. Assume that the point of interest is p and the obstacle is X . Then, using arguments similar to those of Section 2.4.3, the GAC for the first component is a regular GAC after the irrelevant parts of the obstacle have been removed. In this case the obstacle is *intersected* with the ball of radius d centered at p (see Figure 2.11a-b). We denote the GAC of the first component of a bent probe by $D_1(X, \{p\})$. It is assumed that d is known, therefore, for clarity, it is omitted from D_1 .

Notice that D_1 is *exactly* the set of accessible directions for the first component of a bent probe when the second component is ignored. This means that given a direction in $D_1(X, \{p\})$, then a truncated half-line $(0, d)$ that is oriented along this direction and with an endpoint at p will not penetrate X . However, D_1 is only an *upper bound* on the accessible directions of the first component when the whole probe is taken into account, because it is not guaranteed that an accessible second component direction exists for every direction selected from D_1 .

The algorithm to compute D_1 uses a spherical surface in the depth buffer to approximate the intersection of the obstacle with a ball of radius d . This is similar to the algorithm used to compute the ram GAC for a truncated half-line (d, ∞)

but now the depth-buffer acts as a far clipping surface (see right hand side of Figure 2.7). If the depth-buffer is enabled with a “less-than” comparison, then the clipping operation will approximate the intersection of a ball, as needed.

Again, the quality of the approximation depends on the depth-buffer precision, therefore the distance between the far clipping plane and the near clipping plane should be minimized. We assume that the studied point is accessible to the probe’s tip (see Section 2.3). Therefore it must be at a distance of at least r from the obstacle, where r is the radius of the tip. The near clipping plane is then set to a distance of $r/\sqrt{3}$, which is the furthest it can be and still have the viewing volume include all the points that are outside of the tip (see right hand side of Figure 2.7). The far clipping plane is placed at a distance d , which is a tight bound on the ball of radius d .

The clipping operation is only an approximation of the Boolean intersection between the obstacle and the ball. However, since the near clipping plane does not intersect the obstacle (based on the assumption that the tip does not penetrate the obstacle), then we argue that the projection of the clipped obstacle is correct. Section 2.4.3 gives a similar argument using the far clipping plane.

The complexity of computing D_1 , i.e., the GAC of a truncated half-line $(0, d)$, is identical to the complexity of computing the GAC of a truncated probe (d, ∞) , which is the abstraction of a shrunken ram (see Section 2.4.3). Our experiments confirm this fact and show that the cost of computing D_1 is nearly identical to the cost of computing a regular GAC. Figure 2.8d illustrates the D_1 cone with respect to a simple obstacle. Notice that the GAC in Figure 2.8b is the intersection of the cones in Figure 2.8c and Figure 2.8d. This illustrates the fact that an accessible direction of the probe as a whole must be a common accessible direction of its components.

2.5.2 Second Component Accessibility

When the first component takes every possible orientation in D_1 , the articulation point between the first and second components traverses a locus which is the projection of D_1 on a sphere of radius d that is centered at p . Without loss of generality we assume that p is the origin, and we denote this locus by dD_1 , since it is also the result of scaling D_1 by a factor of d .

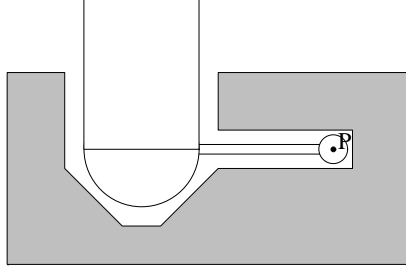


Figure 2.12: A point that is *accessible* but not *approachable* by a bent probe

If we succeed in placing the second component such that the articulation point lies in dD_1 , it is clear that the first component can be placed with its tip at the origin without collisions. In other words, if the second component can access some point of dD_1 , then the entire probe can access the origin. The converse is also true and therefore the origin is accessible iff the second component accesses some point of dD_1 (see Figure 2.11).

The set of directions for which the second component can access *at least one* point of dD_1 is called the *weak* GAC of the feature dD_1 . In general, the weak global accessibility cone of a feature F with respect to an obstacle X , is denoted by $WGAC(X, F)$, and corresponds to the directions from which *at least one* point in F can be accessed by a half-line (notice the analogy to weak visibility [59]). While the GAC of a feature is the *intersection* of the GACs of all the points in the feature, the WGAC of a feature is the corresponding *union*.

We denote the GAC of the second component as D_2 . Then $D_2(X, \{p\}) = WGAC(X, F)$, where the feature F is $D_1(X, \{p\})$ scaled by d about the point p . To compute D_2 we sample points on F and take the union of the GACs at these points. This is a lower bound on the real D_2 , so it is a pessimistic approximation. Figure 2.11c shows F when p is the origin, and illustrates the GAC of a point q sampled on F . This GAC will be a part of the union that forms D_2 .

Notice that the *accessibility* of a point by a bent probe is weaker than the concept of *approachability*. The fact that a bent probe can access a point does not guarantee that there exists a collision-free path for the probe to reach the point. (This happens to be true with *straight* probe abstractions.) Figure 2.12 shows an example of a point that is accessible, but not approachable by a bent probe with the given obstacle. Computing the approachable directions for the second component of a bent probe

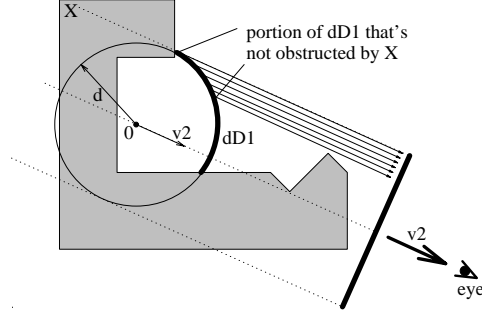


Figure 2.13: Computing $D'_1 \subset D_1$ that corresponds to \vec{v}_2

is a problem that can be as hard as the FindPath problem [37]. We use accessibility instead of approachability, because of the efficient algorithms that are available. A generate-and-test planner, as described in the introduction, will have to verify the approachability condition with a path planner or a simulator. Our experiments on real-world mechanical parts show that failures of this kind occur infrequently.

2.5.3 First Component Accessibility Revisited

We have shown how to compute the GAC of the first component, D_1 , and from this the GAC of the second component, D_2 . The D_2 cones are used to compute setup orientations from which points are accessible to the bent probe. Once a setup is selected, we wish to compute the directions from which the first component of the probe can access a point.

Given a direction of the second component, $\vec{v}_2 \in D_2$, what are the corresponding directions of the first component for the bent probe to access a point p ? Without loss of generality we assume that p is the origin, then the articulation point must lie in dD_1 . Hence, for each $\vec{v}_2 \in D_2$ we are looking for the directions $\vec{v}_1 \in D_1$, such that the second component oriented along \vec{v}_2 and with endpoint at $d\vec{v}_1$ does not collide with X . Spyridi [72] observed that these directions correspond to the points on dD_1 that are not obstructed by X in the orthographic projection of dD_1 onto a plane perpendicular to \vec{v}_2 . Figure 2.13 illustrates this fact. The projection lines in the figure correspond to possible placements for the second component.

We use this observation in our algorithm to compute the subset $D'_1 \subset D_1$ that corresponds to \vec{v}_2 . The viewing parameters for the orthographic projection are depicted in Figure 2.13. We use a parallel projection with direction \vec{v}_2 and a view port large enough to enclose the projection of the ball of radius d , which is a superset of dD_1 . To check if a point on dD_1 is obstructed by the obstacle X we use the depth-buffer in a process that is similar to the two-pass z-buffer shading algorithm [19]. First, we render X into the depth-buffer. Next, we check if a point is obstructed by X by transforming it to the viewing coordinates and comparing its depth value with the value in the depth-buffer. It is not obstructed by X iff its depth value in the appropriate depth-buffer location is closer to the viewer. Note that D_1 is represented by bitmaps on the faces of a cube and therefore dD_1 is also discretized. This is another approximation used by the algorithm. To maximize depth-buffer precision, the distance between the near and far clipping planes should be minimized, while still enclosing the obstacle.

The top of Figure 2.14 illustrates the result of computing D_1 . The length of the probe, d , is equal to the radius of the sphere used to represent the cone. It took 0.07 seconds to compute D_1 . The bottom of the figure shows the accessible directions for the first component of a bent probe, D'_1 , given that the second component is normal to the figure. Notice that these directions are exactly those that are not obstructed by the obstacle in the given view (some of the obstructed direction are inside a slot). It took 0.07 seconds to compute D'_1 . The inaccuracies in the cone are due to aliasing effects from the use of the same low resolution frame buffer (32×32 bits) as with the GAC algorithm and the limited precision available with the depth-buffer. In addition, Figure 2.14 is illustrated with a *perspective* projection rather than an *orthographic* projection, which is used to compute the obstructed directions.

2.6 Summary and Conclusions

This chapter describes simple and efficient algorithms that exploit computer graphics hardware to compute accessibility information for applications in spatial reasoning. Our approach is an unconventional application of graphics hardware. We approximate spherical projections using perspective projections, and we use clipping and the depth-buffer to approximate the intersection and the difference of a solid with a

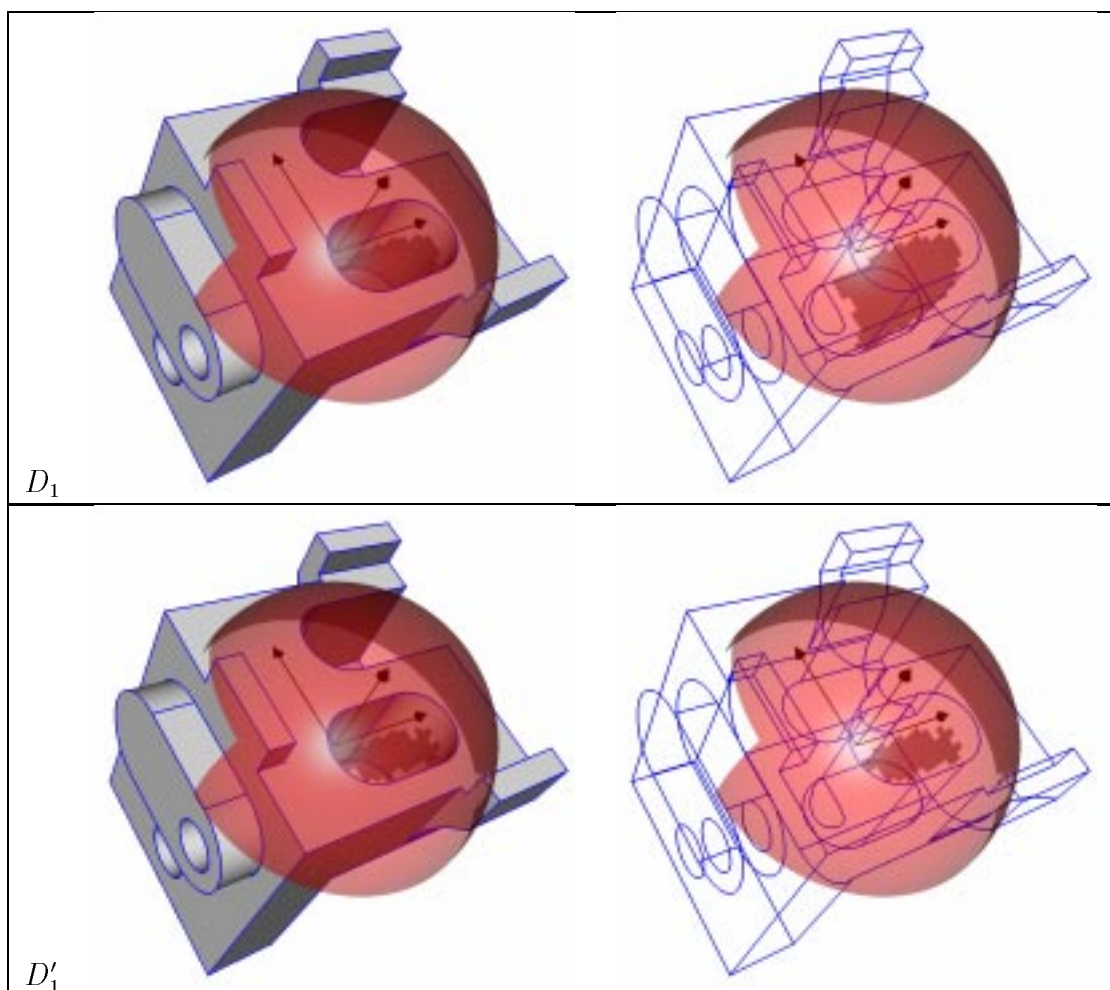


Figure 2.14: Experimental results – D_1 and D'_1

sphere. The depth-buffer is also used to compute the articulation points (between the first and second components of a bent probe) that are not obstructed by an obstacle under an orthographic projection.

The algorithms have been implemented and tested. The empirical results are satisfactory for practical applications with parts of realistic complexity. In the following chapters we will show how to integrate these algorithms into a dimensional inspection planner for CMMs.

Dimensional inspection planners and other spatial reasoning systems must solve complex geometric problems. Exact algorithms for solving these problems often are unavailable, or they are too expensive for real-life applications. Therefore, approximations are unavoidable. Fortunately, the solution spaces of practical planning problems are often large, and optimality (which is usually unattainable with the limited computational resources available) is not required, although the resulting plans must not be grossly suboptimal.

Pessimistic approximations may discard good solutions but they ensure that those found are correct. Optimistic approximations, on the other hand, may include incorrect solutions. And certain approximations may both miss correct solutions and include incorrect ones. The trade-offs between computational complexity and the correctness and quality of the solutions are difficult to assess. It is important to understand that geometric algorithms used in spatial reasoning are not a goal unto themselves. They are meant to be used within a planning framework. Thus, although geometric algorithms that use optimistic approximations may produce invalid results, these can be excluded if a planner includes a verification step. If a verifier fails, the plan must be discarded, and backtracking is needed. Backtracking itself is an expensive proposition, and failures must not occur often.

The algorithms described here use a variety of abstractions and approximations to control the complexity of the computations. A non-exhaustive list of these approximations follows.

- Approachability is approximated by accessibility, which is more tractable. Accessibility implies approachability for straight probes, but not for general bent probes, and therefore the approximation is optimistic.

- The CMM is abstracted as a ram/probe assembly. This ignores possible collisions caused by the remainder of the CMM, and therefore it is optimistic.
- The ram and the probe are further abstracted as grown half-lines. This is a pessimistic approximation if the actual probe is enclosed by the abstraction.
- The GAC of a feature is computed by sampling points from the feature. This leads to an optimistic approximation of the GAC. However, if the inspection planner first selects the sampling points (see Chapter 3) and these are used in the GAC computation, then no approximation is involved.
- The quality of the results depends on the number of pixels of the screens (faces of a cube) used to represent the direction cones. The resolution is a parameter of the algorithm, but higher resolution implies increased computation.

Chapter 3

High-Level Planning

3.1 Introduction

High-level inspection planning involves spatial reasoning, to determine how to orient the part on the CMM, which probes to use, how to orient the probes, and what measurements to perform.

Previous inspection planners are either incomplete [7, 72, 75, 76] or only solve the problem partially, because of the simplifying assumptions used. A common simplification is that the part setup orientation is predetermined [33, 87] or is selected from a handful of orientations [14, 23]. Other systems make simplifying assumptions about the toleranced part — they assume turned parts [17] or parts with a limited set of machining features [54]. In addition, current inspection planners lack coherent definitions for plan quality.

In our work, we develop a general theory for the *existence* of a high-level inspection plan and the requirements for a plan of *high quality*. The inspection planning problem is mapped into a constraint satisfaction problem (CSP) (Section 3.3) and hierarchical constraints are defined to reflect the requirements for a plan of high quality (Section 3.4). A solution to the CSP is a plan, hence a plan exists iff there is a solution to the CSP. We show how to extract approximate solutions to the CSP using efficient clustering methods (Section 3.5).

The planning scheme that we propose is also attractive from a software engineering point of view. We decompose the problem into *knowledge acquisition* and *plan extraction*. Knowledge acquisition involves the computation of domains for the variables in the CSP, and plan extraction involves finding a solution to the CSP.

This is a bottom-up approach that is similar to the feature-based inspection planner [54], but is not limited to specific features. The planner architecture is the topic of Chapter 4.

3.2 Problem Statement

In this chapter we focus on the high-level inspection planning task (see Figure 1.2). Given the model of a specific CMM and the model of a part with design tolerances, we wish to produce a *high-level inspection plan* (HLIP). The HLIP specifies how to setup the part on the CMM, which probes to use, how to orient the probes, and the surface features that are to be measured at these configurations.

3.2.1 Input (CMM and Toleranced Part)

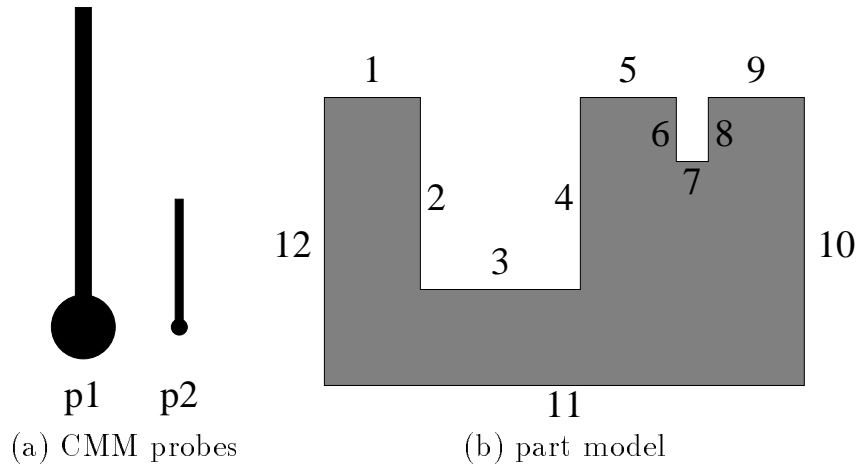
The first input to the inspection planner is a model of a specific CMM. This model consists of the table, ram, and a list of probes that can be attached to the ram (see Figure 3.1a). In our work, we assume that the probes are attached to the ram through a fixed head (e.g., Renishaw PH6) or an orientable head (e.g., Renishaw PH9) [64]. The whole ram/probe assembly corresponds to straight probes or bent probes, respectively, which were introduced in Chapter 2.

The second input to the planner is a model of the toleranced part. This model contains the nominal geometry of the part (Figure 3.1b) with attached attributes that describe the design tolerances. We use a feature-based tolerancing scheme, illustrated in Figure 3.1c.

3.2.2 Measurement Graph and Measurements

The relationships between the entities that define the design tolerances of a part form a *measurement graph* (see Figure 3.2). For now, we do not distinguish between regular features and datum features, but see Section 3.6.2 for special cases.

Each directed path from the root to a face in the measurement graph is called a *measurement*. For example, (T1,F5,4) is a measurement in Figure 3.2. This triplet can be interpreted as “inspect face 4 as a component of feature F5 to check tolerance



Feature	Description	Faces
F1	top	1, 5, 9
F2	left	12
F3	right	10
F4	bottom	11
F5	large-slot	2, 4
F6	small-slot	6, 8
Datums	Description	Ftrs
D1	top	F1
Tolerance	Description	Ftr/Dtm
T1	perpendicularity	F5, D1
T2	perpendicularity	F6, D1

(c) design tolerances

Figure 3.1: Example input to the inspection planner

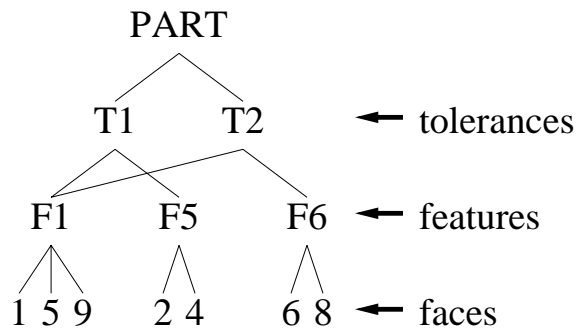


Figure 3.2: The measurement graph

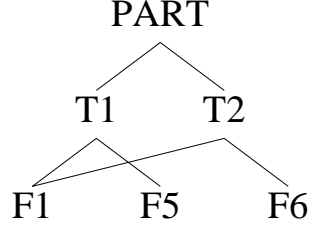


Figure 3.3: The simplified measurement graph

T1". We will see that the context (i.e., feature, tolerance) in which a face is measured is important.

We use a wild card notation to define certain groups of measurements. For example, in Figure 3.2, $(T2, F1, *)$ denotes the three measurements: $(T2, F1, 1)$, $(T2, F1, 5)$, and $(T2, F1, 9)$. This is an example of a *same-prefix* measurement group. The measurement group $(*, *, *)$ denotes all the measurements. The depth of a same-prefix measurement group is defined to be the length of the prefix. Hence, $(*, *, *)$ is of depth 0, and $(T2, F1, *)$ is of depth 2. Symmetrically, we define *same-suffix* measurement groups. The same-suffix groups of depth 1 are of particular importance, because they denote all the measurements that are associated with a specific face. For example, $(*, *, 5)$ includes measurements $(T1, F1, 5)$ and $(T2, F1, 5)$.

To avoid unnecessary complexity in the remainder of this chapter, we remove the faces from the tolerance specification. Therefore, the measurement graph is now composed of 3 levels, as in Figure 3.3, and measurements are composed of tolerance-feature pairs.

3.2.3 Output (HLIP-tree)

The output of the high-level inspection planner is a HLIP. We represent a collection of HLIPs using a tree structure of depth 5, called a *HLIP-tree*. The nodes at depth 1 represent part setups, at depth 2 probes, at depth 3 probe orientations, at depth 4 features, and at depth 5 measurements. Equivalently, one can interpret the nodes as arguments to the following operators:

- **change-setup** — change the setup of the part on the CMM table, including fixtures and clamps.

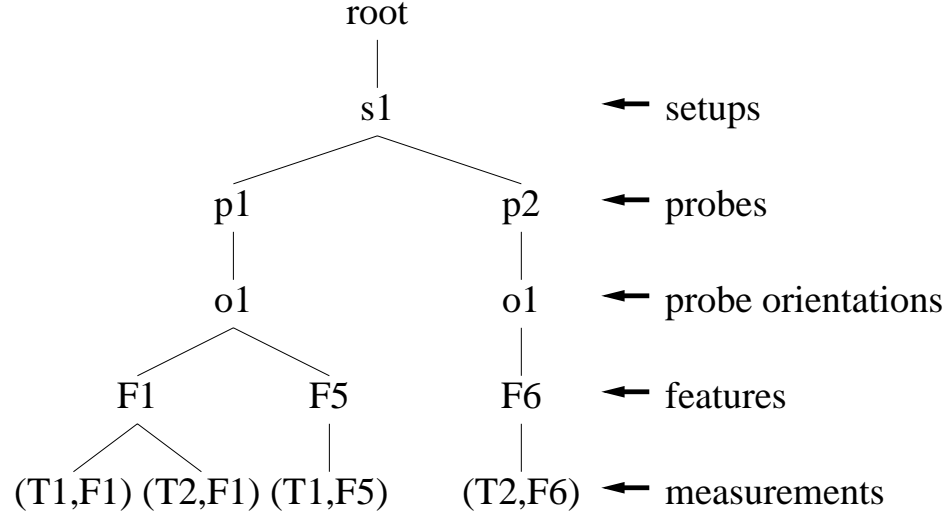


Figure 3.4: A HLIP-tree

- **change-probe** — change the probe that is attached to the CMM ram.
- **change-orien** — change the orientation of the probe relative to the CMM. This operator does not apply to CMMs with fixed heads.
- **inspect-ftr** — inspect a feature by physically moving the CMM ram and sampling points on the surface. The actual path plan for the CMM ram are low-level details to be discussed in Chapter 5.
- **record-meas** — records that a specific measurement has been performed. This is a book-keeping mechanism that keeps track of which measurements have been accomplished.

The first 4 operators change the physical state of the machine, while the last operator is a book-keeping mechanism that does not affect the execution of the plan. Notice that the inspection of a single feature may accomplish several measurements. Typically, if a feature F is inspected, then measurements $(*,F)$ are accomplished.

Figure 3.4 illustrates a HLIP-tree for the input of Figure 3.1, where $s1$ is the setup that is depicted in Figure 3.1b, the probes are indexed as in Figure 3.1a, and the probe orientation $o1$ is the vertical orientation in which the probe is aligned with the ram. Notice that the **inspect-ftr** operator is really an **inspect-face** operator if the original measurement graph is used.

The HLIP-tree is a representation for a family of linear plans, which correspond to a depth-first traversal of the tree. For example, using a depth-first left-to-right traversal of the tree in Figure 3.4, we produce the following linear plan:

- 1) `change-setup(s1)`
- 2) `change-probe(p1)`
- 3) `change-orien(o1)`
- 4) `inspect-ftr(F1)`
- 5) `record-meas(T1,F1)`
- 6) `record-meas(T2,F1)`
- 7) `inspect-ftr(F5)`
- 8) `preform-meas(T1,F5)`
- 9) `change-probe(p2)`
- 10) `change-orien(o1)`
- 11) `inspect-ftr(F6)`
- 12) `record-meas(T2,F6)`

Other plans are produced by changing the order in which nodes at the same level are visited. This representation for plans was chosen to reflect plan quality considerations, as explained in Section 3.4 below. Linearizing a HLIP-tree is considered low-level planning and is the topic of Chapter 5. Notice that additional operators are needed for a complete plan, for example, an operator that translates the CMM ram is needed to inspect a feature. However, these are also considered low-level details that will come into effect in Chapter 5.

A HLIP is *valid* if it is physically feasible to inspect all the features (i.e., perform the `inspect-ftr` operators) under the specified setup, probe, and probe orientation. In other words, each feature must be *approachable* by the CMM in the given configuration (see Figure 2.12). Furthermore, a measurement can only be recorded after an appropriate feature has been inspected. The goal of the HLIP is to perform all the measurements at hand (plan quality is discussed in Section 3.4).

3.3 Planning by Constraint Satisfaction

A *constraint satisfaction problem* (CSP) is formally a set of variables and a set of constraints [35, 68]. A solution to a CSP is an assignment of values to the variables that do not violate any constraints. In this section we first formulate the high-level inspection planning problem as a CSP and then show how to construct a HLIP-tree to represent solutions.

3.3.1 Variables (Measurements)

The variables in our CSP are the measurements to be performed as specified by the measurement graph. The domain of values that a measurement can have is the set of all tuples of the form $(\mathbf{s}, \mathbf{p}, \mathbf{o})$, where \mathbf{s} is a setup, \mathbf{p} is a probe, and \mathbf{o} is a probe orientation. The idea is that when we assign a tuple $(\mathbf{s}, \mathbf{p}, \mathbf{o})$ to a measurement $\mathbf{M}=(\mathbf{T}, \mathbf{F})$, then we wish to perform measurement \mathbf{M} when the part is in setup \mathbf{s} using probe \mathbf{p} at orientation \mathbf{o} . In other words, we wish to inspect feature \mathbf{F} in this configuration, and record that measurement \mathbf{M} has been accomplished.

3.3.2 Constraints (Approachability)

To obtain valid HLIPs we need a unary constraint, which specifies the allowable subset of the domain that can be assigned to each variable. In our case, the variables must satisfy the *unary approachability constraint*. A tuple $(\mathbf{s}, \mathbf{p}, \mathbf{o})$ satisfies the unary approachability constraint for a measurement $\mathbf{M}=(\mathbf{T}, \mathbf{F})$ if the feature \mathbf{F} can be inspected in setup \mathbf{s} with probe \mathbf{p} at orientation \mathbf{o} (so that the measurement \mathbf{M} can be performed). Ensuring approachability is not trivial, but practical approximation techniques exist (e.g., accessibility analysis).

Binary constraints between two variables specify the allowable subset of the cross-product of the two domains. Such constraints are not needed to ensure valid plans, but will be introduced in Section 3.4 to generate plans of high-quality.

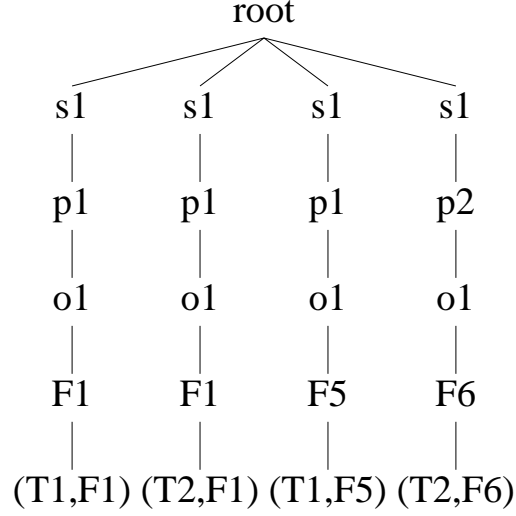


Figure 3.5: A trivial HLIP-tree

3.3.3 Existence of Solutions

The CSP we defined is trivial, in the sense that there are only unary constraints. Therefore, a solution exists iff we can assign a value to each variable independently of the rest. In other words, a solution exists iff all the allowable subsets of the domains (after applying the unary approachability constraint) are non-empty.

If a solution exists, then we simply assign an arbitrary value to each variable from the allowable subset of its domain. However, computing this subset is non-trivial, and the topic of Chapter 4.

3.3.4 Constructing Plans from CSP Solutions

A solution to the CSP is trivially converted into a non-optimized HLIP-tree, as depicted in the example of Figure 3.5. These trees can be optimized by merging common nodes [20], to produce HLIP-trees as in Figure 3.4. In our case, the merging process is a simple matter of lexicographically ordering the values assigned to the variables. Notice, however, that since the assignment to each variable is independent of others, then the merged plan may still be of poor quality.

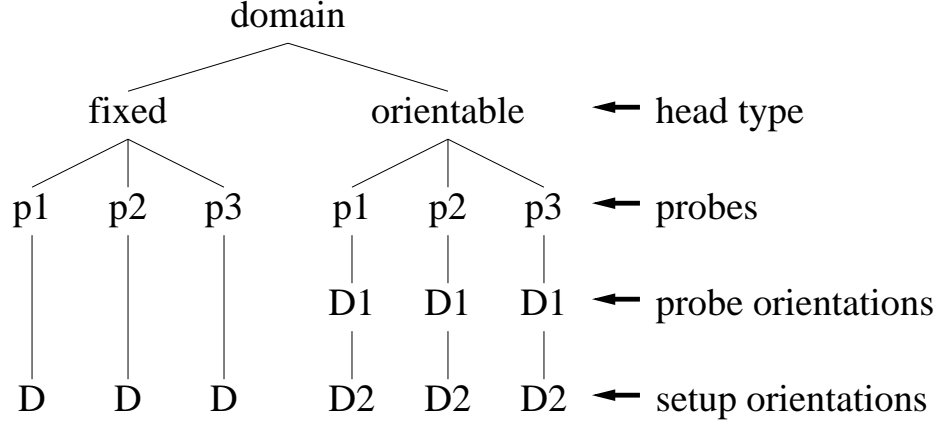


Figure 3.6: Representing the domain of a measurement

3.3.5 Representing Domains

In Section 3.3.1, we defined the domain of values that a measurement can have as the set of all tuples of the form $(\mathbf{s}, \mathbf{p}, \mathbf{o})$, where \mathbf{s} is a setup, \mathbf{p} is a probe, and \mathbf{o} is a probe orientation. Notice that the unary approachability constraint is a purely geometric constraint, which only depends on the feature specified by the measurement and nothing more (datums are an exception, and the topic of Section 3.6.2). In other words, each measurement in a same-suffix measurement group of depth 1 (or greater) will have an identical allowable subset of the domain. For this reason, it makes sense to store the domain of variables per feature instead of storing the domains per measurements. This obviously saves storage space but also time of computation, because the unary constraints are not applied for each measurement.

In the spirit of Chapter 2, the representation for the domain of variables depends on the type of ram head in use. For fixed heads (e.g., Renishaw PH6), we called the whole ram/probe assembly a *straight probe*, and for orientable heads (e.g., Renishaw PH9), we called the whole ram/probe assembly a *bent probe*. Figure 3.6 illustrates the domain representation. For fixed heads, we store a single direction cone that specifies the valid setup orientations for each probe in use. For orientable heads, we store a pair of direction cones that correspond to valid probe orientations and setup orientations, respectively.

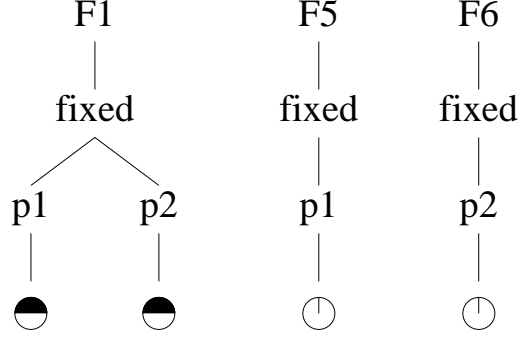


Figure 3.7: The allowable values for the measurements in Figure 3.1

Accessibility analysis is used to approximate the unary approachability constraint in a computationally practical manner. For straight probes, the valid setup orientations are approximated by the global accessibility cone (see Section 2.4). For bent probes, the probe orientations and setup orientations relative to the part are approximated by the (D_1, D_2) pair of direction cones (see Section 2.5).

Notice that the direction cones are stored for each probe. This is necessary, because each probe may produce different accessibility cones. The advantage of this scheme is that better probe approximations can be used during accessibility analysis. The number of available probes is typically small, so the additional complexity is not large. In addition, if a probe is found unsuitable for inspecting a specific feature (e.g., during tip accessibility analysis (see Section 2.3) or if one of the direction cones is empty), then it can be pruned from the domain.

Figure 3.7 illustrates the allowable values for measurements in the example of Figure 3.1. For simplicity, we assume that the CMM has a fixed head. Notice that each slot can only be inspected with a single probe — probe **p1** is too large for slot **F6** and probe **p2** is too short for slot **F5**. In addition, each slot can be inspected from a single setup orientation (the one illustrated in Figure 3.1), while the datum **F1** can be inspected from a hemi-sphere of setup orientations.

A domain object should support the following methods (we assume that the CMM head type, i.e., fixed or orientable, is known in advance):

- Compute all possible setup orientations. This is the union of all the direction cones that correspond to setup orientations.

- Given a setup orientation, compute all possible probes. If the setup orientation is taken from the set of all possible orientations as computed above, then the set of possible probes is non-empty.
- Given a setup orientation and a probe, compute all possible probe orientations. This method is only relevant to orientable heads (i.e., bent probes). In this case, we locate the D_1 cone for the given probe and compute D'_1 for the specified setup orientation (see Section 2.5.3 for details). If the setup orientation and probe are taken from the sets computed above and (D_1, D_2) are computed as in Section 2.5, then the D'_1 cone is non-empty (see [71] for detailed proof).

A value can be assigned to each variable that has a non-empty domain. The value is computed in the following manner. First, compute all possible setup orientations and select one. Then, compute all possible probes for the specific setup orientation and select one. Finally, if the head is orientable, then compute D'_1 for the specific probe and setup orientation, and select a probe orientation. Since the computed sets are always non-empty, we are guaranteed not to fail.

3.4 Plan Quality and *Same* Constraints

The previous section showed how the inspection planning problem can be encoded as a CSP. However, a plan that satisfies only the approachability constraint is not likely to be of high quality. This section focuses on plan quality. The goal is to encode the requirements for a good plan as hierarchical constraints, so that a solution to the hierarchical CSP is a HLIP of high quality.

The quality of an inspection plan is largely determined by three factors:

- **Efficiency** in terms of the time it takes to execute the plan. We wish to maximize efficiency in order to minimize the time it takes to execute the plan.
- **Accuracy** in terms of the errors involved during the inspection of a tolerance. We wish to obtain satisfactory accuracy (not necessarily maximum accuracy) in order to verify that the features are within the tolerances specified by the designer.
- **Cost** in terms of the hardware equipment used (e.g., expensive probes).

The cost of probes can be encoded as preferences and will be discussed in Section 3.6.1. In this section we emphasize the first two factors — efficiency and accuracy of high-level inspection plans. We believe that a precise analysis of the cost (in terms of efficiency and accuracy) of each high-level operator, is not practical. Instead, we use a more qualitative analysis. The most expensive operation is a change of setup, which typically is a time-consuming, manual procedure. Setup changes also are the major cause of positional uncertainties. Changes of probe are the next most expensive and inaccurate operations. Probe orientation is the least expensive and inaccurate. This is the reason for the hierarchy in the HLIP-tree. Inspecting a feature takes time and has a finite accuracy, but these factors are constant from the point-of-view of high-level plans.

Spyridi and Requicha introduced *same* constraints to address plan quality issues [72, 75, 76]. A group of measurements under the **same-setup** constraint must be performed under the same setup in the final plan. A group of measurements under the **same-probe** constraint must be performed with the same probe *and* setup. Finally, a group of measurements under the **same-orient** constraint must be performed under the same setup, probe and probe orientation. In terms of HLIP-trees, the **same-setup** constraint applied to a group of measurements implies that these measurements have a common setup ancestor in the HLIP-tree (assuming that the tree has been merged as in Section 3.3.4). Similarly, the **same-probe** constraint implies a common probe ancestor, and the **same-orient** constraint implies a common probe orientation ancestor. We use the wild card notation, such as **same-setup**(T,*), to denote that the **same-setup** constraint is applied to all measurements in the (T,*) same-prefix measurement group.

Applying *same* constraints, however, can quickly over-constrain the CSP. For example, it may not be possible to perform all the measurements of the same tolerance under the same setup. However, a solution can still be found by minimizing the amount of constraints that are violated. This is the idea behind soft constraints, or over-constrained systems [29]. To ensure that the most important constraints are less likely to be violated, we apply *all* the *same* constraints in a hierarchical manner and form a constraint hierarchy [4].

It is convenient to view the *same* constraints as binary constraints. A *same* constraint that is applied to a group of measurements, such as **same-setup**(T,*),

is a set of binary constraints between every pair of measurements in $(T,*)$. Then, even if **same-setup** $(T,*)$ cannot be satisfied as a whole, we still wish to minimize the number of implied binary constraints that are violated.

In the remainder of this section we describe three constraint hierarchies. The first specifies efficient plans, the second specifies plans of high accuracy, and the third is a combination of the two. Extracting solutions to the third CSP is the topic of Section 3.5.

3.4.1 A Constraint Hierarchy for Plan Efficiency

We make the assumption that changing the setup is much more time consuming than changing the probe, which is much more time consuming than changing the probe orientation. Therefore, we obtain the following constraint hierarchy:

```
H0 { approachability(T,F) : forall tolerance T, feature F }
H1 { same-setup(*,*) }
H2 { same-probe(*,*) }
H3 { same-orient(*,*) }
```

H0 are the required constraints in the system. We require the unary approachability constraint, otherwise we are not guaranteed that solutions will be valid HLIPs. The remainder of the constraints are soft constraints. A solution to the hierarchical CSP should satisfy the required constraints H0 and minimize the number of violated soft constraints in a lexicographical order, i.e., a solution should violate the least amount of H1, and then the least amount of H2, and so forth. (In the constraint hierarchy theory, this is called the *locally-predicate-better* valuation of a solution [4].)

The most restrictive constraint is H3, i.e., **same-orient** for all measurements. This implies that all measurements are to be performed under the same configuration of setup, probe and probe orientation. Obviously this is the ideal but it is rarely satisfied in practice.

Notice that by definition if H3 is satisfied then so are H2 and H1. It therefore seems redundant to specify these two additional constraints in the hierarchy. However, this is precisely what we want, because if H3 cannot be satisfied, then perhaps H2 can be, and so forth.

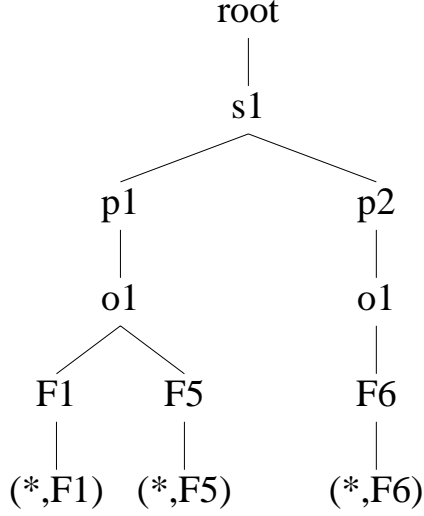


Figure 3.8: A HLIP-tree representing efficient plans

Figure 3.8 illustrates a solution to the hierarchical CSP given above. This HLIP-tree is identical to the one in Figure 3.4 except that the leaves are specified as same-suffix measurement groups of depth 1. This was done to illustrate the fact that when a feature is inspected, then all measurements associated with it can be performed, i.e., `same-orien(*,F)` is always satisfied. In effect, we automatically minimize the number of `inspect-ftr` operators in the HLIP, because there is exactly one of each feature in the tree. In the following section, we will see that there are cases when a feature is inspected more than once to obtain plans of higher accuracy.

3.4.2 A Constraint Hierarchy for Plan Accuracy

The constraint hierarchy for plan accuracy is defined as:

```

H0 { approachability(T,F) : forall tolerance T, feature F }
H1 { same-setup(T,*) : forall tolerance T }
H2 { same-probe(T,*) : forall tolerance T }
H3 { same-orien(T,*) : forall tolerance T }

```

This is very similar to the constraint hierarchy for plan efficiency given in the previous section. For accuracy we are concerned with minimizing the errors when performing a group of measurements under each tolerance, as opposed to minimizing the number of global operators.

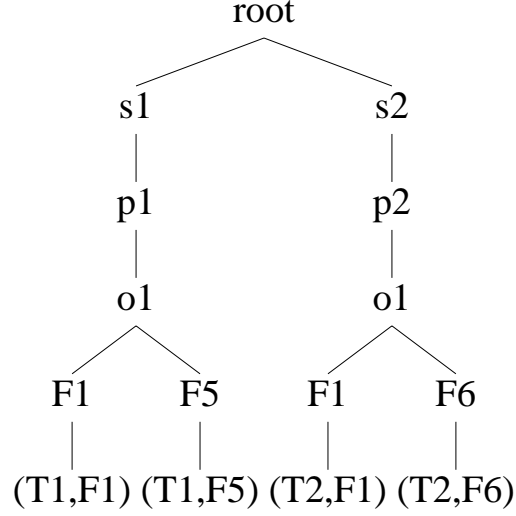


Figure 3.9: A HLIP-tree representing plans of high accuracy

Figure 3.9 illustrates a solution to the constraint hierarchy. In this example we assume that the CMM has an orientable head and that **s2** is the setup that has the part resting on face number 12 (see Figure 3.1). This is not an efficient plan, because two setups are used and feature **F1** is inspected twice. However, it is an accurate plan (by our definition), because all the measurements under each tolerance are performed with the same setup, probe and probe orientation, i.e., $(T1,*)$ and $(T2,*)$ each have a common orientation ancestor in the HLIP-tree. In other words, none of the constraints are violated.

For completeness, we supply the constraint hierarchy for the non-simplified measurement graph that contains faces as well (see Figure 3.2):

```

H0 { approachability(T,F,C) : tolerance T, feature F, face C }
H1 { same-setup(T,F,*) : forall tolerance T, feature F }
H2 { same-setup(T,*,*) : forall tolerance T }
H3 { same-probe(T,F,*) : forall tolerance T, feature F }
H4 { same-probe(T,*,*) : forall tolerance T }
H5 { same-orient(T,F,*) : forall tolerance T, feature F }
H6 { same-orient(T,*,*) : forall tolerance T }

```

The idea is that if a feature can be inspected with minimal errors, then this gets precedence over the entire tolerance.

3.4.3 Efficient Plans of High Accuracy

In this section we wish to combine plan accuracy and plan efficiency and define a constraint hierarchy that will produce good plans. Both constraint hierarchies that were defined are not satisfactory on their own. We have shown that not every accurate plan is an efficient plan. The reverse is also true. For example, the efficient plan of Figure 3.8 violates the **same-orien**(T2,*) and **same-probe**(T2,*) constraints in the constraint hierarchy for plan accuracy.

For inspection we are much more concerned with the accuracy of the plan than the efficiency of the plan as long as the generated plan is practical. Generating a plan that does not produce accurate results will be useless in the inspection process. We propose to interleave the constraint hierarchies giving precedence to accuracy and obtain the following hierarchy:

```
H0 { approachability(T,F) : forall tolerance T, feature F }
H1 { same-setup(T,*) : forall tolerance T }
H2 { same-setup(*,*) }
H3 { same-probe(T,*) : forall tolerance T }
H4 { same-probe(*,*) }
H5 { same-orien(T,*) : forall tolerance T }
H6 { same-orien(*,*) }
```

Figure 3.10 illustrates a solution to this hierarchical CSP. The constraints, specified in H2, H4, and H6, eliminate solutions that are not efficient. For example, the HLIP-tree in Figure 3.9 is not a solution in this case, because H2 is violated.

Notice that if changing the orientation of a probe does not introduce serious positional uncertainties into the system (with respect to the tolerances that are to be inspected), then the **same-orien** constraints may produce far from optimal plans at no gain of practical accuracy. Therefore, in this case, H5 should be eliminated. It is important to have the flexibility to modify the constraint hierarchy to suit the needs of the specific problem.

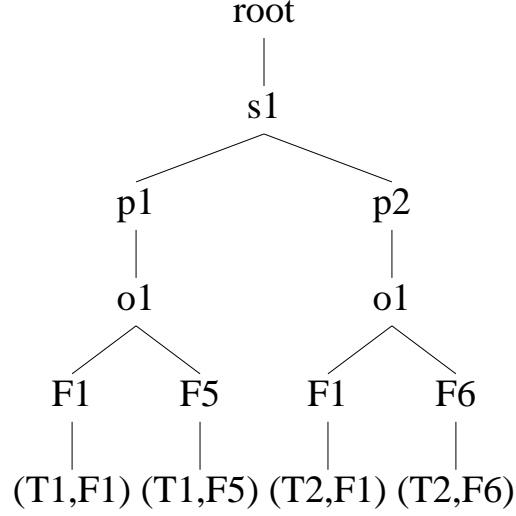


Figure 3.10: A HLIP-tree representing efficient plans of high accuracy

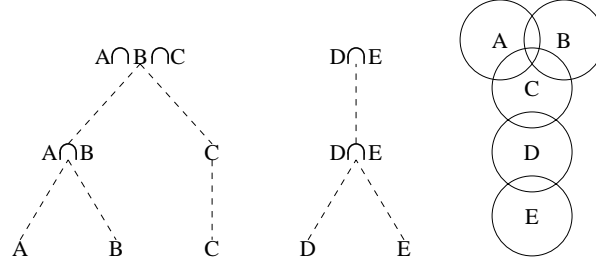


Figure 3.11: The clustering operation

3.5 Hierarchical Constraint Satisfaction

We propose a greedy method that exploits the structure of the hierarchical CSP to find approximate solutions to the problem. The method uses clustering techniques, which can be applied directly to the domains of measurements.

This section is outlined as follows. First we define a valid clustering operation, the process of clustering up the measurement graph, and measurement sub-graph extraction. Then we describe the main constraint satisfaction algorithm, and solve two example problems. We conclude with an efficient algorithm that performs a valid clustering operation.

3.5.1 Clustering Up the Measurement Graph

Let $\{\omega_i\}$ be a collection of sets. (In our application the ω_i are subsets of the domains of the CSP variables.) A *cluster* C is a collection of ω sets whose intersection is non-empty. A cluster $C = \{\omega_1, \dots, \omega_k\}$ has an *underlying set* μ , which is the non-empty intersection of its constituent sets, i.e., $\mu = \bigcap_{i=1}^k \omega_i$. Given the ω_i 's we construct a set of clusters C_j such that:

1. Each ω_i belongs to one and only one C_j .
2. The underlying sets of the clusters are pairwise disjoint, i.e., $\mu_i \cap \mu_j = \emptyset$ for all $i \neq j$.

We call the process of partitioning the ω_i 's into clusters (property 1) a *clustering* operation. If property 2 is satisfied, then we call it a *valid clustering* operation. An algorithm that performs a valid clustering operation is given in Section 3.5.6.

The clusters that result from a valid clustering operation cannot be merged into larger clusters. If we attempted to merge two clusters, C_i and C_j , the result would be a collection of ω sets with empty intersection because $\mu_i \cap \mu_j = \emptyset$, and therefore not a cluster.

Figure 3.11 illustrates the successive application of clustering operations. Given the sets A – E shown on the right, they can be clustered to yield $\{A, B\}$, $\{C\}$ and $\{D, E\}$, which satisfy property 1 but not property 2, because $\{A, B\}$ and $\{C\}$ can be merged into $\{A, B, C\}$. Now, we consider the underlying sets of these clusters, $A \cap B$, C and $D \cap E$, and apply a *valid* clustering operation. In the graph on the left in the figure, we represent a cluster by its underlying set plus links to the cluster's constituent sets.

Clustering up the measurement graph is a bottom-up process of assigning a set of clusters to each node of the graph. Leaf nodes are assigned a set containing a single *primitive cluster* itself composed of a single set. If the children of a non-leaf node have been assigned sets of clusters, then we consider the underlying sets of all the clusters involved and apply a valid clustering operation to them. The result is assigned to the parent node. The clusters assigned to the root node, PART, are called *top clusters*.

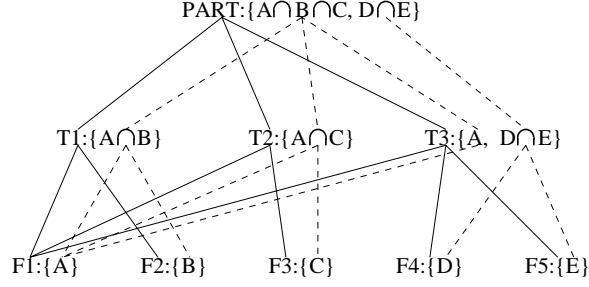


Figure 3.12: Clustering up a measurement graph

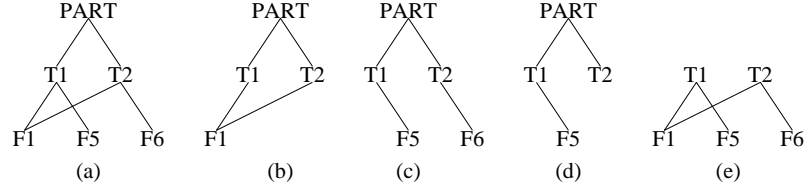


Figure 3.13: Sub-graphs and measurement sub-graphs

Figure 3.12 shows an example of clustering up a measurement graph, assuming the same original sets of Figure 3.11. The arcs in the measurement graph are drawn with solid lines, while the clusters' links are drawn with dashed lines.

3.5.2 Measurement Sub-Graph Extraction

A *measurement sub-graph* is a sub-graph of the measurement graph that has the structure of a measurement graph of lower complexity. For example, Figure 3.13 shows 5 sub-graphs for the measurement graph (a), shown on the left. However, only (a), (b), and (c), are measurement sub-graphs. The two on the right do not have the appropriate structure: (d) has a tolerance node with no children, and (e) has no root node.

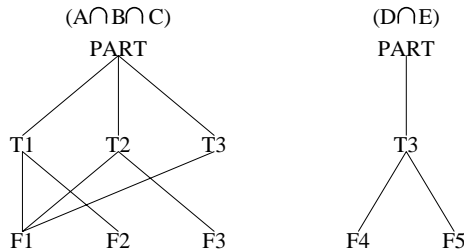


Figure 3.14: Measurement sub-graph extraction

As a result of clustering up the measurement graph the root node is assigned a set of top clusters, as shown in Figure 3.12. Notice that each top cluster corresponds to a measurement sub-graph. (Each cluster link has a corresponding arc in the measurement graph.) Figure 3.14 shows the two sub-graphs that correspond to the top clusters in Figure 3.12.

Measurement sub-graph extraction is the process of generating the measurement sub-graph that corresponds to a particular top cluster. By traversing the links through the descendents of a top cluster, it is easy to identify the arcs and nodes that are included in the measurement sub-graph. This is the core of a measurement sub-graph extraction algorithm.

3.5.3 Main Constraint Satisfaction Algorithm

In this section we describe the hierarchical constraint satisfaction algorithm. We wish to solve the CSP given in Section 3.4.3. The input is a measurement graph and the domains of variables stored at the leaves of the graph. The domain includes the set of allowable $(\mathbf{s}, \mathbf{p}, \mathbf{o})$ tuples (see Section 3.3.5). It is assumed that the unary approachability constraint is already computed for each domain, but the algorithm will work with under-constrained domains as well. If the approachability constraint is approximated, then a verification step is needed (see Chapter 4). Notice that all the domains must be non-empty, otherwise there is no solution to the CSP. If all the domains are non-empty, then there exists a solution and our algorithm will find one, but it may be sub-optimal.

The main algorithm extracts a HLIP-tree from the measurement graph as follows:

1. First, assign to each leaf node a set containing a single primitive cluster that represents the allowable setup orientations for inspecting this face. Next, cluster up the measurement graph, and for each top cluster, C_s , do the following:
2. Select a setup $s \in C_s$, and extract the measurement sub-graph that corresponds to C_s . Assign to each leaf node a set containing a single primitive cluster that represents the allowable probes under setup s . Cluster up the measurement sub-graph, and for each top cluster, $C_{s,p}$, do the following:

3. Select a probe $p \in C_{s,p}$, and extract the measurement sub-graph that corresponds to $C_{s,p}$. If the CMM has a fixed head, then we are done. Otherwise, assign to each leaf node a set containing a single primitive cluster that represents the allowable probe orientations under setup s and probe p . Cluster up the measurement sub-graph, and for each top cluster, $C_{s,p,o}$ do the following: select an orientation $o \in C_{s,p,o}$, and extract the measurement sub-graph that corresponds to $C_{s,p,o}$.

Each $(\mathbf{s}, \mathbf{p}, \mathbf{o})$ tuple generated by the algorithm represents a branch in the HLIP-tree. Under each branch we have a measurement sub-graph that corresponds to $C_{s,p,o}$. A leaf in this measurement sub-graph corresponds to a feature, \mathbf{F} , that is to be inspected in configuration $(\mathbf{s}, \mathbf{p}, \mathbf{o})$. The same-suffix measurement group $(*, \mathbf{F})$ corresponds to the measurements that are accomplished when the feature is inspected.

The end of Section 3.3.5 explains how to create the primitive clusters that are assigned to the leaf nodes. In particular, how to extract from the domain of a variable: (1) the allowable setup orientations, (2) the allowable probes under a given setup, and (3) the allowable probe orientations under a given setup and probe.

It is easy to see the symmetry in the algorithm. After clustering up the measurement graph we select the configurations (either setups, probes, or probe orientations). Setups are selected to optimize H1 and H2, then probes are selected to optimize H3 and H4, and so forth. In each case, the measurement graph is decomposed into sub-graphs. The resulting HLIP-tree will have *disjoint* sets of measurements under each probe orientation. The sets are disjoint, because as a result of clustering up the measurement graph each measurement corresponds to one and only one branch in a single top cluster. (This is derived from the fact that each leaf is assigned a set with a *single* primitive cluster, and from the properties of a valid clustering operation.)

This algorithm generates an approximate (i.e., sub-optimal) solution, because each configuration is selected independently of the following ones. For example, we select a minimum set of setups, but a different minimum set may be better in the sense that fewer probes may be needed. Figure 3.15 shows two possible setups, but the setup on the left requires two probes to inspect the highlighted features.

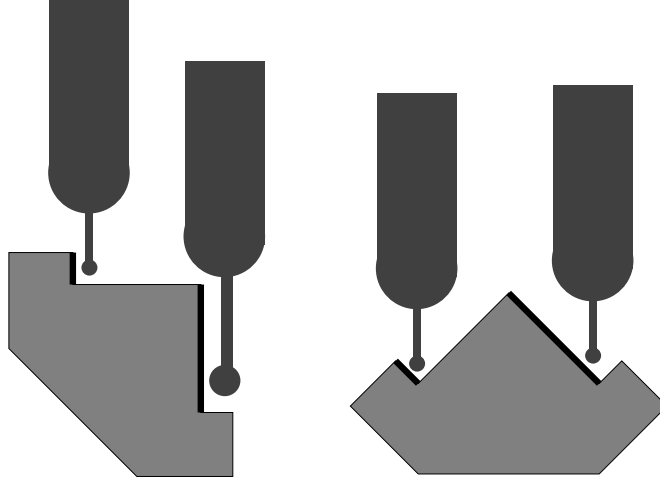


Figure 3.15: Probe selection depends on setup selection

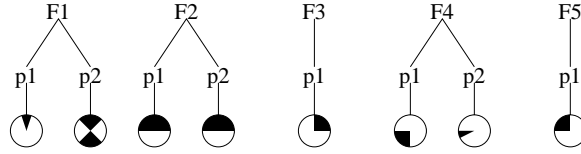


Figure 3.16: Example domains (CMM has a fixed head)

(The large probe is too big for the small feature, and the small probe cannot access the large feature under the setup illustrated on the left.) The setup on the right is better, because the small probe can inspect both features. Unfortunately, our algorithm clusters the setups without regard to the probes, so the setup on the left is just as likely to be selected as the setup on the right. Setup preferences can be used to avoid such situations (see Section 3.6.1).

In addition, we cluster up the measurement graph in a greedy fashion, and do not guarantee to find the minimum number of top clusters (see Section 3.5.6).

3.5.4 First Example

In this section, we trace the constraint satisfaction algorithm for a toy problem. For simplicity, we assume that the CMM head is fixed (in this case we do not have to compute probe orientations explicitly, since they are the same as the setup orientations), and two probes are available, $p1$ and $p2$. In addition, we do not provide

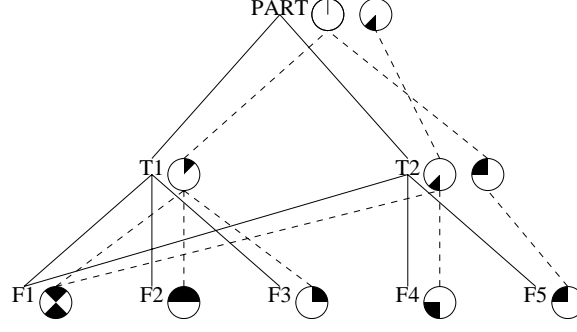


Figure 3.17: Clustering the setup orientations

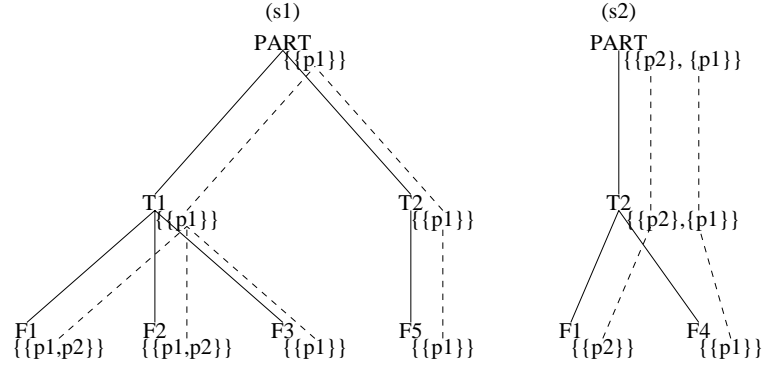


Figure 3.18: Clustering the probes

the geometry of the part or probes. All we need are the domains of the variables, Figure 3.16, and the measurement graph, Figure 3.17. The domains include the 2-D GACs for each feature with respect to each probe. A missing probe (in the domains of **F3** and **F5**) implies that the respective GAC is empty.

Step 1 in the algorithm clusters the setup orientations, i.e., GACs. Each leaf node **F** is assigned a set with a single primitive cluster. This cluster represents the union of the GACs in the domain stored in **F**. Next, we cluster up the measurement graph as shown in in Figure 3.17. Two top clusters are formed. We select one setup from each, say **s1** (up direction) and **s2** (down direction), and extract the corresponding measurement sub-graphs. The sub-graphs appear in Figure 3.18.

In the next step, for each measurement sub-graph, we cluster sets of probes. Again, each leaf node **F** is assigned a set with a single primitive cluster. This time, the cluster represents the set of probes that can access **F** under the given setup orientation **s**. To compute the set of probes, we classify **s** against the GAC under

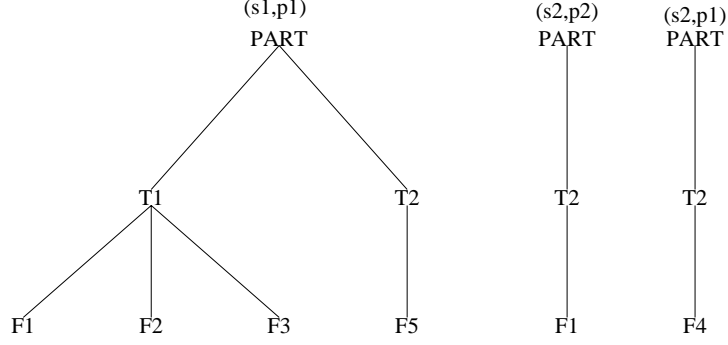


Figure 3.19: The measurement sub-graph for each (s,p) configuration

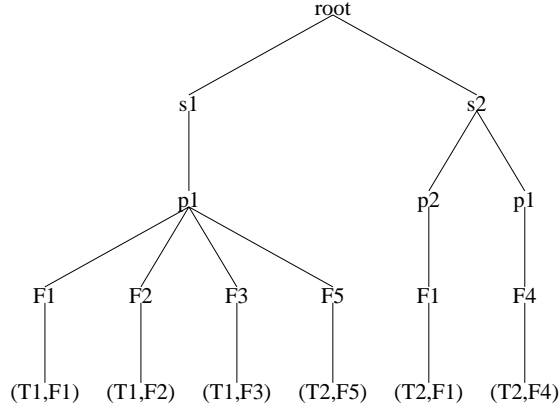


Figure 3.20: The resulting HLIP-tree

each probe p in the domain stored in F . If s is a member of the GAC under p , then p can access F under setup orientation s . For example, $s2$ is not a member of the GAC under $p1$ in the domain stored in $F1$. Therefore, the primitive cluster in the set assigned to $F1$ in the measurement sub-graph that corresponds to $s2$ (on the right of Figure 3.18) is $\{p2\}$, and does not include $p1$.

We cluster the probes up the measurement sub-graphs, as in Figure 3.18. The graph on the left has a single top cluster, while the graph on the right has two top clusters. Again, we select a single probe from each top cluster, and extract the sub-graphs that are shown in Figure 3.19.

We assume that the CMM head is fixed, so we are done with the main algorithm. We arrange the results in the form of a HLIP-tree as shown in Figure 3.20. Each (s,p) pair that was generated is a branch in the HLIP-tree. The measurement sub-graphs in Figure 3.19 appear under each branch, but in a different format.

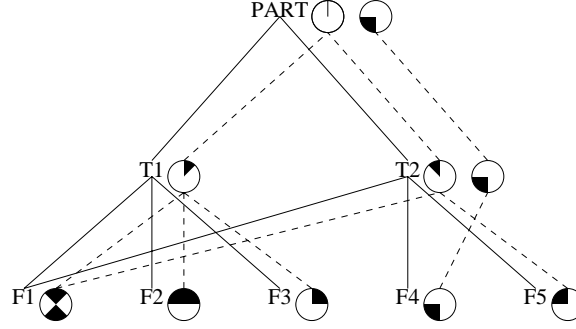


Figure 3.21: Clustering the setup orientations differently

First, we enumerate the leaves of the sub-graphs, i.e., the features \mathbf{F} that are to be inspected under configuration (\mathbf{s}, \mathbf{p}) . Then we enumerate the measurements that are accomplished when the feature is inspected, i.e., $(\mathbf{*}, \mathbf{F})$ with respect to the sub-graph.

Notice, that the generated HLIP-tree in this example is not the most efficient, because feature $\mathbf{F1}$ is inspected twice. It is the most accurate in the terms we defined in Section 3.4.2. However, in this example, there is no real accuracy gained by inspecting $\mathbf{F1}$ twice if the tolerance $\mathbf{T2}$ cannot be inspected entirely under the same setup. A better clustering of the setups would have produced the desired result. Specifically, if the set of clusters assigned to $\mathbf{T2}$ were as depicted in Figure 3.21 as opposed to the clusters in Figure 3.17. The result depends on the implementation of the valid clustering operation, which is discussed in Section 3.5.6.

3.5.5 Second Example

Now, we trace the algorithm for the example introduced at the beginning of this chapter. The input is given in Figure 3.1. To avoid unnecessary complexity, we assume that the CMM head is fixed and we use the simplified measurement graph of Figure 3.3

The domains of the measurements are illustrated in Figure 3.7. The clustering of the setup orientations is shown in Figure 3.22a. The clustering is trivial, and produces a single top cluster. We select a setup orientation, $\mathbf{s1}$, from the top cluster and extract the corresponding measurement sub-graph. In this case, the sub-graph is the entire graph. Next, we cluster the probes, as shown in Figure 3.22b. The result has

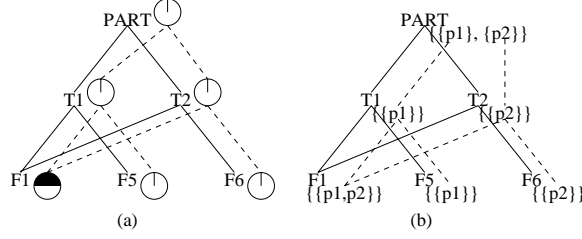


Figure 3.22: Clustering (a) setup orientations, and (b) probes

two top clusters. We select one probe from each, and decompose the measurement graph into two sub-graphs (as illustrated by the dashed lines in Figure 3.22b).

In this example, we assume that the CMM has a fixed head, and therefore do not select probe orientations. The result HLIP-tree, illustrated in Figure 3.10, is the desired solution to the constraint hierarchy.

3.5.6 Clustering Algorithm

We conclude this section with an algorithm that performs a valid clustering operation, as defined in Section 3.5.1. Notice that a valid clustering operation does not necessarily produce a minimum number of clusters. The *minimum clustering problem* has been shown to be NP-hard [72]. We propose an algorithm that efficiently produces a set of clusters that may not be minimal in size, but is close to minimal in practice.

The following pseudo-code performs a valid clustering operation on a collection of sets $W = \{\omega_i\}$. The result is a set of clusters $\mathcal{A} = \{C_j\}$. Ω denotes a set that contains all the sets in W . If the ω_i denote direction cones (i.e., setup orientations or probe orientation), then Ω is a complete direction cone. If the ω_i denote sets of probes, then Ω is the set of all possible probes.

1. $\mathcal{A} \leftarrow \emptyset$
2. Shuffle W (optional)
3. While $W \neq \emptyset$ do the following:
 - (a) $C \leftarrow \emptyset$ /* the new cluster to be */
 - (b) $\mu \leftarrow \Omega$ /* the underlying set of C */

- (c) For each $\omega \in W$ do the following:
 - i. $\mu' \leftarrow \mu \cap \omega$
 - ii. if $\mu' \neq \emptyset$ then do the following:
 - A. $W \leftarrow W \setminus \{\omega\}$
 - B. $C \leftarrow C \cup \{\omega\}$
 - C. $\mu \leftarrow \mu'$
- (d) $\mathcal{A} \leftarrow \mathcal{A} \cup \{C\}$

The main loop (line 3) constructs clusters from the sets in W and inserts them into the collection of clusters, \mathcal{A} . The inner loop (line 3c) incrementally constructs C from the remaining sets ω in W . As long as the intersection $\mu \cap \omega$ is not empty, $C \cup \{\omega\}$ is a cluster, so we safely add ω to C and remove it from W . It is clear that each ω_i in W is inserted into one and only one C_j in \mathcal{A} , so the algorithm is a clustering operation (property 1 in Section 3.5.1). In addition, the greedy construction of a cluster C guarantees that the underlying set μ does not intersect any of the remaining sets ω in W . This implies property 2, i.e., $\mu_i \cap \mu_j = \emptyset$ for all $i \neq j$, so the algorithm is a valid clustering operation.

Consider again the sets A – E shown on the right of Figure 3.11. If the order of the input is $W = \{A, D, B, E, C\}$, then the algorithm will first produce cluster $\{A, B, C\}$ followed by $\{D, E\}$, and then terminate. The result is a minimum number of clusters. However, if the input sequence is $W = \{C, D, A, B, E\}$, then the algorithm will generate the set $\mathcal{A} = \{\{C, D\}, \{A, B\}, \{E\}\}$, which is not a minimum number of clusters. As you see, the clustering algorithm is sensitive to the order of the input. For this reason we provide an optional step to shuffle W in line 2. In this way we provide a *chance* to obtain a set of clusters of minimal size (but see Section 4.2 for the drawbacks of using non-determinism).

The clustering algorithm is greedy in the way we construct the next cluster C , but it is not greedy in the usual sense. That is, the algorithm does not select at each iteration a cluster C with the most number of ω_i . If it did, then the output would not depend on the order of the input. In any case, such a scheme would be more time consuming, and as far as we know does not guarantee better results.

In the worst-case, the algorithm performs $O(n^2)$ intersections, where n is the size of the input set, W . This is clear, because the main loop may traverse n times if

only one ω is removed from W at each iteration. This will happen if all the sets in W are disjoint, i.e., $\omega_i \cap \omega_j = \emptyset$ for all $i \neq j$. It is obvious that *at least one* ω is removed at each iteration, because of the way μ is initialized in line 3b.

In the best-case, all the elements of W have a common intersection. Then W is a cluster in itself, and a valid clustering operation *has to* produce $\mathcal{A} = \{W\}$. In this case the algorithm iterates through the main loop once and performs a total of n intersections in the inner loop. In practice, we expect the number of clusters in the result, \mathcal{A} , to be bound by some fixed number. Therefore, for practical purposes the algorithm performs a linear number of intersections.

3.6 Closing Loose Ends

In the process of developing our inspection planning theory we have over-simplified certain details. This section covers three important issues that should be considered during the high-level planning process and have been integrated into our planner: preferences (for setups, probes, and probe orientations), replacement of datum features by mating surfaces, and feature segmentation.

3.6.1 Preferences for Setups, Probes and Orientations

In our definition of good plans (see Section 3.4.3) we assume that setup changes are more expensive (and more error prone) than probe changes, which are more expensive than orientation changes. We explained why this is a reasonable assumption. However, by this definition, all setups have the same cost, all probes have the same cost, and all probe orientations have the same cost. This assumption may be true for probe orientations, but it is not true for probes that have varying costs, and it is not true for setups — some setups are preferred for the ease of fixturing, stability and other considerations.

Ranking probes by their cost is trivial. Identifying preferred setups is not the focus of this dissertation. Preferred setups can be identified by an external agent, such as a stability analysis module or a human operator. In our planner, we assume that prioritized lists of preferred setups and probes are available. The challenge is to integrate these preferences into the hierarchical constraint satisfaction algorithm of

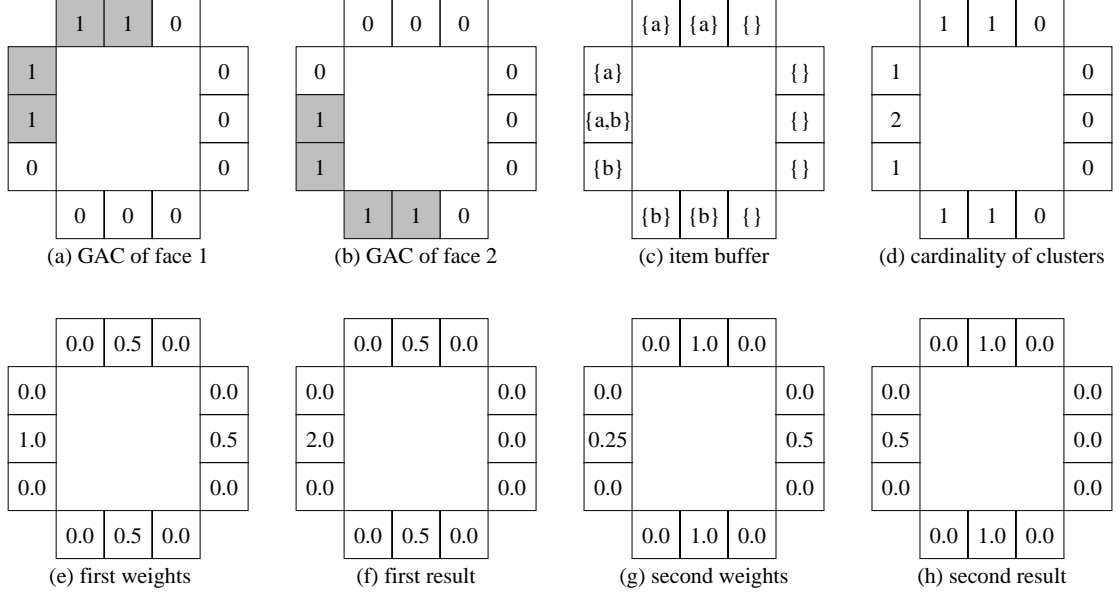


Figure 3.23: Cluster selection based on a weight function

Section 3.5. For the remainder of this section we focus on setup preferences. Probe and probe orientation preferences are handled in an analogous manner.

The simplest way to integrate the setup preferences is during the setup selection (step 2 in the algorithm). Instead of selecting an arbitrary setup from each top cluster, setups are selected based on their priority. This is the approach we took, but it has its drawbacks. Notice that the setup selection occurs *after* clustering up the measurement graph. Therefore, preferred setups can get lost in the clustering process, which knows nothing about them.

Spyridi [72] proposed that preferences be integrated into the clustering process *before* the selection is done. The idea is to ensure that the preferences are not violated by the clustering process. In particular, she proposed that the greedy clustering algorithm (Section 3.5.1) select clusters that maximize a weight function that depends on the size of the cluster and the setups that it includes. Notice that this is only a heuristic (the minimum clustering problem is still NP-hard), and there may be better ways to do this.

In our planner, we represent direction cones with cubic maps (Section 2.4.1). Therefore, it makes sense to use an *item buffer* [19] to compute all possible clusters for a set of direction cones. The idea is to have each pixel store (an ID of or

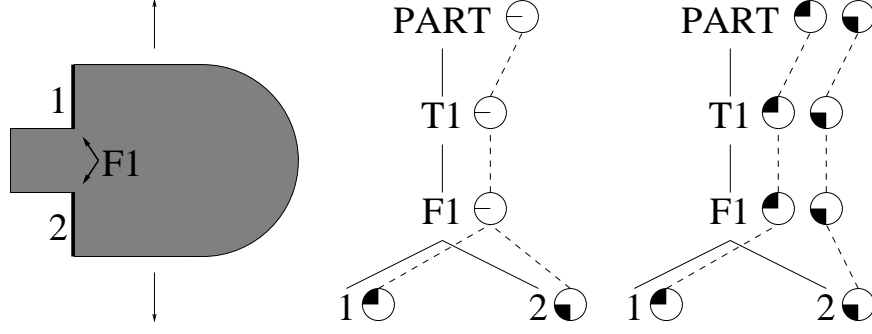


Figure 3.24: A single unstable setup vs. two preferred setups

reference to) the cluster that includes all the cones with the corresponding pixel set to 1. Then, the best cluster can be evaluated as a function of its size and the weights of the setup orientations in the underlying set. For simplicity, we consider the function that is the product of the cluster size and the weight assigned to the underlying pixel. Figure 3.23 gives a concrete example. Let (a) and (b) be the cubic mappings of 2-D direction cones. The item buffer for these is given in (c). The size of the clusters is shown in (d). The weights assigned to the setup orientations is given in (e), where 1.0 is a strong preference and 0.0 is a weak preference. The result of pair-wise multiplication of the elements in (d) and (e) is shown in (f). The best cluster in this example is $\{a, b\}$, which has a weight of 2.0. Similarly, the result of multiplying (d) and (g) is shown in (h), and there is a tie between two best clusters, $\{a\}$ and $\{b\}$, each having the weight of 1.0.

Notice that the weights can produce very different inspection plans. For example, Figure 3.24 illustrates a part with a single feature to be inspected, F1 (the union of faces 1 and 2). For simplicity, we consider a half-line probe, and the trivial measurement graph shown in the figure. The GACs of the highlighted faces are attached to the leaf nodes. Notice that they correspond to the cubic mappings in Figure 3.23 (a) and (b). Therefore, clustering up the measurement graph with the weights specified in (e) will produce the left clustering in Figure 3.24, while the weights in (g) will produce the right clustering. In the former case, the clustering algorithm generates a single top cluster with a single setup that has the round face on the table. This is clearly not a desirable setup unless a suitable fixture is available. The latter case, gives considerable weight to two setups — one with the bottom face resting on the table and the other with the top face resting on the table. The result

has two top clusters each containing one of the preferred setups. In this case, feature F1 is not inspected in a single setup, but no fixtures are needed to hold the part. Without fixturing information, the planner cannot determine which inspection plan is better. It is important to have the flexibility to modify the weights as needed.

Notice that the clustering operation is no longer *valid*. For example, the underlying sets in the top clusters on the right side of Figure 3.24 are no longer disjoint. We could force the clustering operation to be valid by removing common, but undesirable directions. A similar technique can be used to guide our planner, which does not use a weight function: by removing undesirable directions we can avoid undesirable clusters.

3.6.2 Replacement of Datum Features by Mating Surfaces

Datum features are defined as ideal surfaces that are associated in a specific manner with physical part features [3]. For example, a planar datum is “tangent” to the corresponding part surface. This implies that we can inspect a planar datum surface by placing the physical surface against the CMM table and inspecting the table. This observation generalizes to other datum surfaces that mate with fixturing surfaces.

In our planner, we have integrated the possibility of placing planar datum features on the CMM table. To do so, we have modified the domain representation for measurements of datums and the measurement sub-graph extraction algorithm. The original domain representation includes all the configurations from which a feature can be inspected (Section 3.3.5). When a datum is placed on the table, we inspect the table and not the datum feature. Therefore, the configurations from which the table can be inspected, for a given datum feature, are stored in a separate domain structure. The domain of the datum is then the *union* of the original domain and the special domain just defined.

Figure 3.25 illustrates the domain representation when the datum is placed on the CMM table. There is a unique setup orientation, and we assume that the table is large enough, so that all the probes can inspect it. For fixed heads we do not need to store the probe orientations, since the probe must be aligned with the CMM ram. For indexable heads we limit the possible probe orientations, so that the ram does not penetrate the table. The possible probe orientations depends on the length of

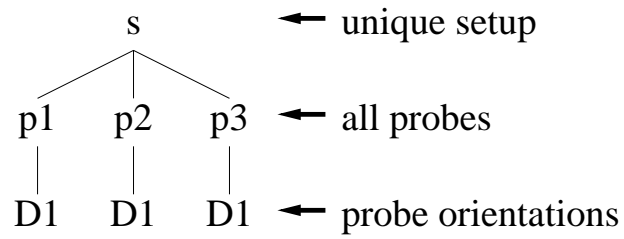


Figure 3.25: Representing the domain of a datum that is placed on the table

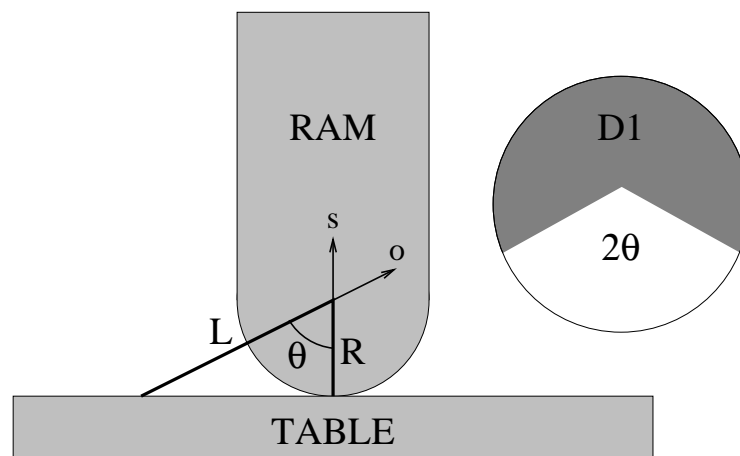


Figure 3.26: Valid probe orientations for inspecting the table

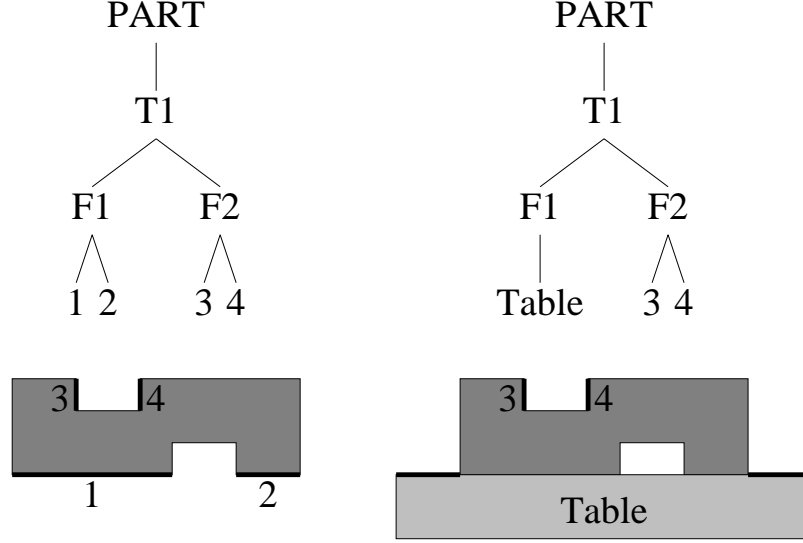


Figure 3.27: The measurement sub-graph for a setup that places a datum on the table

the probe, L , and the radius of the ram, R — the dot product between the direction of the ram, \mathbf{s} , and the direction of the probe, \mathbf{o} , should not exceed R/L (Figure 3.26).

In the constraint satisfaction algorithm we assign primitive clusters to the leaf nodes before clustering up the measurement graph. If a leaf node corresponds to a datum, then the original domain is considered combined with the new domain. In this manner, the setup that places a datum on the table is united with other possible setups. After clustering up the measurement graph, the setup that places the datum on the table may be selected from a top cluster. If this is the case, then care is taken during measurement sub-graph extraction. The measurement sub-graph should reflect the fact that the datum feature itself is not to be inspected, rather the table that it is resting on. This is accomplished by replacing the datum's children by a symbolic face that represents the CMM table. For example, the left side of Figure 3.27 illustrates a simple measurement graph, where **F1** is a planar datum that can be placed on the table. If after clustering setups we select the setup that places the datum on the table, then the measurement graph is transformed to the one on the right of the figure.

Notice that datums are replaceable by the table in a specific context. For example, a datum has to be measured directly in order to inspect a flatness tolerance. In this case we treat the datum as a regular feature. If the same feature has to be

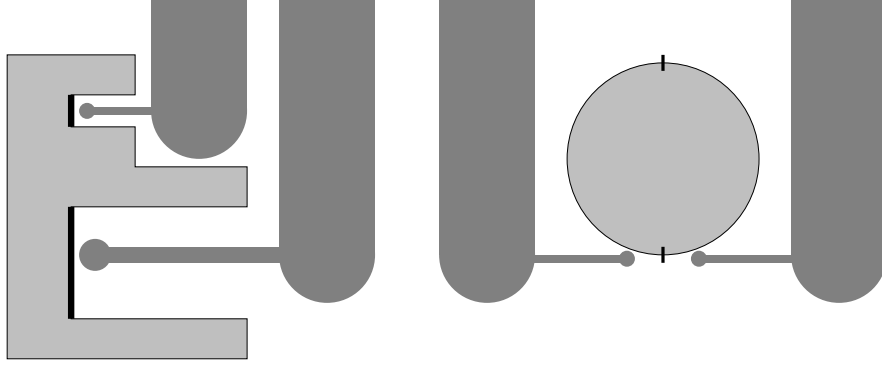


Figure 3.28: Segmentation examples

treated in two different ways, the easiest solution is to instantiate the feature twice in the measurement graph.

3.6.3 Segmentation and Point Sampling

This section addresses the problem of measurements that cannot satisfy the unary approachability constraint. This implies that there is no solution to the CSP, but does this really mean that no suitable inspection plan exists? The answer is no, because we have inadvertently over-constrained the problem. For example, in some cases a feature is not approachable as a whole, but individual segments of the feature are. The process of segmenting a feature into sub-features is called *feature segmentation*, and is a known problem in high-level inspection planning [72].

Figure 3.28 shows examples of features that are not approachable as a whole. The simplest example is a spherical surface feature. A sphere has an empty GAC and has to be approached from at least two directions, as shown on the right hand side of Figure 3.28. On the left we show a feature that requires both of the probes shown.

Notice that some features have no approachable points. Others may have a strict subset of points that are approachable. Actually, this is the typical case for pockets and slots that have “hard to reach” corners. Therefore, even if we do segment a feature into infinitely small patches, some may not be approachable. We need some criteria for selecting a suitable *candidate set* of patches that can be inspected to

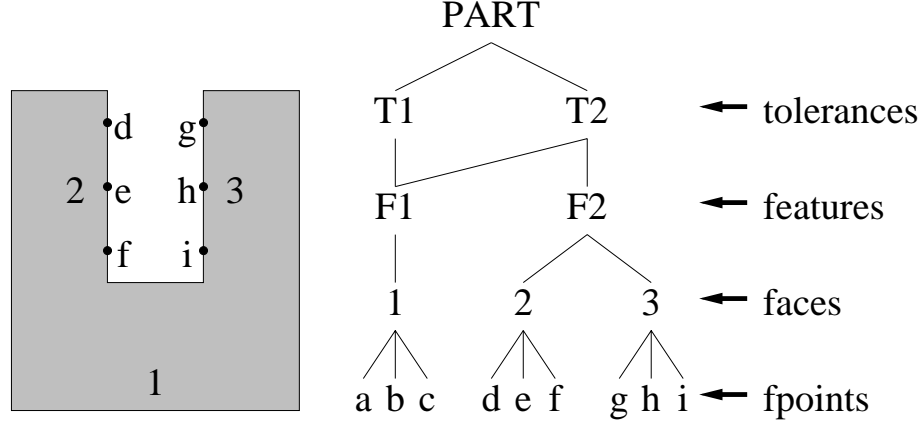


Figure 3.29: The measurement graph with fpoints sampled from each face

measure the feature as a whole, or to determine that no such candidate set exists, which implies that the feature cannot be inspected and no inspection plan exists.

Instead of using infinitely small patches, we can approximate them by points on the surface of the feature. This has several advantages: our accessibility analysis algorithms are most efficient for single points, and CMMs typically inspect discrete points on the surface. The candidate set of points on the feature can be the actual points to be inspected by the CMM (or a superset of them).

Measurement point selection is a branch of research in itself [11, 88, 39] and not the focus of this thesis. Previous researchers have attempted to separate the measurement-point selection from high-level planning and apply it as low-level refinement of the high-level plan [72]. This makes the problems of feature segmentation and high-level plan validation more difficult to solve.

In our scheme, we propose to sample points on each face, and to use them to represent the surface. We call such points *fpoints*. For each fpoint we store its position on the face and the normal to the surface. The measurement graph is augmented by an extra level of detail as in Figure 3.29, and the domains of measurements are now stored per fpoint in the system. Similarly, an `inspect-fpnt` operator replaces the `inspect-face` operator. All the algorithms remain the same with an extra layer of complexity. There is one exception in that fpoints are *filtered* before the clustering process, because we only need a candidate set of fpoints. Specifically, we filter out fpoints that do not have a large set of possible probes in their domains. Without

this filtering process, a poorly selected fpoint, such as one located near a tight corner of a slot, will have strong influence on the outcome of the clustering process. In an extreme case, the domain associated with an fpoint may be empty, and the point should definitely be filtered out. The idea of filtering out fpoints is similar to filtering out noise in a system (see Section 4.1.1 for details).

Currently, we sample points uniformly over the surface of the feature. In practice, the distribution may depend on the geometry of the surface, or even on how the feature was manufactured. More general techniques for measurement point selection can be integrated into the planner, but this is not the focus of this dissertation.

Chapter 4

Planner Architecture

4.1 Main Algorithm

So far we have mapped the inspection planning problem to a CSP and showed how to extract good plans in the form of a HLIP-tree. In a sense, we decomposed the problem into *knowledge acquisition* and *plan extraction*. Knowledge acquisition involves the computation of domains for the variables in the CSP, i.e., satisfying the unary approachability constraint through accessibility analysis (Chapter 2). Plan extraction involves finding a solution to the CSP, i.e., hierarchical constraint satisfaction through clustering techniques (Chapter 3).

A naive planner can compute *all* the domains and then extract a HLIP-tree. This design is especially suitable to a highly parallel architecture, since each domain can be computed independently of the others. Unfortunately, it is not practical to compute the domains *precisely*. Instead we use approximations, such as accessibility analysis, and represent the domains with discrete direction cones. These approximations typically are optimistic, and therefore the planner needs a verification step to validate the plans. If the plan is invalid, then better approximations must be used.

In practice, knowledge acquisition involves expensive geometric computations, and parallel processing is not widely available. Furthermore, plan extraction is relatively cheap and can be performed multiple times with incomplete knowledge. Therefore, knowledge is acquired incrementally when needed, through *lazy evaluation*. Our planner follows the *generate-and-test* paradigm. The idea is to extract a plan and verify it. If the verification step fails, then incrementally acquire more

knowledge and repeat the process. Such a planner is *complete* if the knowledge acquisition is optimistic and converges to the true domains.

The main algorithm is outlined below:

1. Initialize knowledge base.
2. Loop forever or until resources expire (e.g., time limit):
 - (a) If no solution exists, then report failure and stop.
 - (b) Extract a plan.
 - (c) If the plan is valid, then report success and done.
 - (d) Perform incremental knowledge acquisition.

The following sub-sections describe the various steps in the algorithm excluding plan extraction, which was the topic of Section 3.5. The remainder of this chapter discusses possible speed-ups, points of failure and the simulator used in our system. We conclude with experimental results and a discussion.

4.1.1 Knowledge Base Initialization

The knowledge base stores the input to the planner — a model of the CMM and a model of the part. This is *global knowledge* that remains constant throughout the planning process. A preprocessing step extracts relevant information from this input. This includes the measurement graph and other global knowledge, such as preferred setups and probes.

The measurement graph is initialized from the tolerance specification that comes with the part. An additional step filters out tolerances and features that are not appropriate for CMM inspection, such as threaded features. This is a rule-based system that uses purely symbolic reasoning. The user is notified about the entities that cannot be inspected. Before proceeding, the user has the option to edit the measurement graph.

Local knowledge is stored per face or at the leaf nodes of the measurement graph. Each face that is to be inspected is sampled by a set of fpoints (see Section 3.6.3), and at each fpoint we store the domain of allowable values (see Section 3.3.5). The

domains are initialized to contain *all* possible $(\mathbf{s}, \mathbf{p}, \mathbf{o})$ configurations (i.e., setups, probes, and probe orientations). This is the most optimistic approximation possible, but the idea is to incrementally constrain the domains as needed (see Section 4.1.4).

To avoid starting from a completely unconstrained system, we chose a relatively cheap constraint to apply during the initialization step. We perform tip accessibility analysis (see Section 2.3) for all the fpoints in the system. Tip accessibility provides a reliable and efficient mechanism for sensor planning. For example, it prunes large probes from the domains of narrow slots. In addition, poorly selected fpoints are filtered out at this stage. These are fpoints that are accessible to a small number of tips relative to other fpoints in the same face. The idea is to eliminate fpoints that are close to corners and other obstacles.

4.1.2 Testing Existence of a Solution

For each face to be inspected, we check that there is a candidate set of fpoints that have a non-empty domain. As described in Section 3.6.3, the candidate set must meet the criteria of a measurement point selection strategy [11, 88, 39], which is out of the scope of this dissertation. We require that a minimum number of fpoints be uniformly distributed over the face.

4.1.3 Plan Validation

The validation step verifies that the HLIP-tree represents a family of plans that are feasible in practice. In particular, each fpoint that is inspected must be *approachable*. In this section, we focus on the geometric problem of approachability.

We call each $(\mathbf{s}, \mathbf{p}, \mathbf{o}, \mathbf{f})$ branch in the HLIP-tree a *primitive inspection plan* (PIP). We say that a PIP $(\mathbf{s}, \mathbf{p}, \mathbf{o}, \mathbf{f})$ is *valid*, if fpoint \mathbf{f} is approachable when the part is in setup \mathbf{s} using probe \mathbf{p} at orientation \mathbf{o} . A HLIP-tree is valid if all the PIPs in the tree are valid. The approachability constraint guarantees that a valid high-level plan can be refined to a low-level plan (see Chapter 5).

Testing approachability for arbitrary probes is as hard as the regular path planning problem. Instead, we test for accessibility, which can be done efficiently through collision detection. Recall that any approachable PIP is also accessible. The reverse is not always true, but our experiments show that it is rare to encounter such cases.

In any event, PIPs that are accessible but not approachable will be detected during path planning (see Chapter 5).

Notice that we test the accessibility of the entire approach/retract path of an fpoint (see Section 2.4.5). We compute the sweep of the ram/probe assembly along this path and then test for collisions with the part. The ram and probe are abstracted by grown lines (i.e., cylinders that are “capped” by spheres), so the sweep calculations are trivial.

The verification step returns all the invalid PIPs in the HLIP-tree. If no invalid PIPs are found, then the high-level plan is considered potentially valid, and is expanded by the low-level planner (which may fail, since we do not guarantee approachability at the high-level stage).

4.1.4 Incremental Knowledge Acquisition

The verification step returns a set of invalid PIPs. We make sure that these PIPs are not encountered again by excluding them from the knowledge base. Specifically, if a PIP $(\mathbf{s}, \mathbf{p}, \mathbf{o}, \mathbf{f})$ is invalid then $(\mathbf{s}, \mathbf{p}, \mathbf{o})$ is removed from the domain attached to fpoint \mathbf{f} . For straight probes this amounts to removing the direction \mathbf{s} from the setup orientations, D , associated with probe \mathbf{p} (see Figure 3.6). For bent probes we do not represent the (\mathbf{s}, \mathbf{o}) pairs explicitly (through a 4-D cone), so we cannot exclude (\mathbf{s}, \mathbf{o}) without losing possibly valid configurations. To alleviate this problem, with each domain we store the last setup \mathbf{s} , probe \mathbf{p} and D'_1 , that was computed during the plan extraction. Then, the orientation \mathbf{o} is removed from D'_1 . If D'_1 becomes empty, then \mathbf{s} is removed from the D_2 direction cone under \mathbf{p} . (We will see in Section 4.2 that storing D'_1 also offers considerable speed-ups during plan extraction.)

After removing the invalid PIPs, we constrain the domains through accessibility analysis. We have a variety of accessibility analysis algorithms that can be applied with successively better approximations at higher computational cost. For example, half-line abstractions vs. grown half-line abstractions, and accessibility of a single fpoint vs. accessibility of an entire approach/retract path. Originally, our planner applied these algorithms successively, but we soon discovered that the best approximations were performed anyway. This was due to the fact that the clustering process caused the top clusters to be typically small and on the boundaries of its descendant

clusters. Therefore, directions were chosen from these boundaries, which were often accessible directions for half-line abstractions but not for grown half-lines.

For straight probes we compute the intersection of 4 GACs — one GAC for the stylus abstraction (a grown half-line), one GAC for the ram abstraction (a grown truncated half-line), each of these computed both at the point of contact and at the retraction point (i.e., at 2 points on the approach/retract path).

Computing D_2 for bent probes is expensive and depends on the number of points sampled from the D_1 cone. Our experiments show that most often D_2 is nearly complete (i.e., it contains almost all possible setup orientations). Due to this fact, we first compute the D_1 cones and postpone the computation of D_2 . Associated with each domain is a counter that stores the number of times the domain was constrained. This counter is used to decide when to apply each constraint. If the counter reaches a specified limit, then the domain is marked as empty and can be edited by the user.

For bent probes we compute the intersection of two D_1 cones the first time around — one at the point of contact and the other at the retraction point. We use the stylus abstraction of a grown truncated half-line. The next time an invalid PIP from the same domain is encountered, we compute the D_2 cones.

Notice that the invalid PIP $(\mathbf{s}, \mathbf{p}, \mathbf{o}, \mathbf{f})$ contains a specific probe \mathbf{p} . Therefore, we can compute the accessibility cones for the specific probe in the domain (see Figure 3.6). However, our experiments showed that we are better off computing the cones for *all* the probes in the domain. This is another tradeoff between the higher computational cost and the quality of the approximation.

4.2 Speed-Ups

4.2.1 Controller

The key to successfully generate inspection plans within a reasonable amount of time is to acquire the cheapest knowledge that provides the most constraints on the system. This implies ranking the invalid PIPs and the possible accessibility analysis algorithms. A controller would be needed to schedule the knowledge acquisition based on available resources. We did not attempt to design a sophisticated controller

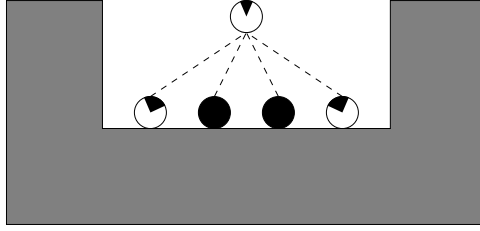


Figure 4.1: Domains of neighboring fpoints

in this dissertation. Instead we chose a very simple one that is discussed below. Again, we point out that knowledge acquisition can be performed in parallel, so in theory all the domains can be constrained at once.

Our controller simply constrains the domain of the first invalid PIP encountered during the verification step. This introduces two speed-ups. First, the verification step can abort when the first invalid PIP is found. Second, once you constrain the domain at one fpoint, then you typically do not need to constrain the domains of *neighboring* fpoints, i.e., fpoints that belong to the same face. The idea is that the domains of neighboring fpoints are the first to be clustered up the measurement graph (see Section 3.5.1), and typically belong to a single cluster. The underlying set represents the intersection of the domains, so by constraining the domain of one fpoint we are implicitly constraining the domains of neighboring fpoints.

For example, Figure 4.1 shows a part with a single slot. Four fpoints are sampled from the bottom of the slot. For sake of illustration, the probe is a half-line and the domain is a single direction cone that represents possible setup orientations. The GAC is computed for the two extremal fpoints. (In the figure, the GACs are centered at the fpoints.) The domains of the two internal fpoints are not constrained, and remain complete direction cones. The result of clustering these direction cones is a set of directions from which *all* the fpoints are accessible, and not just the original two. Of course, this is an ideal example, but it occurs frequently in practice.

4.2.2 Caching Information

Other gains in computational speed are obtained from caching information. As we mentioned in Section 4.1.4, each domain (for a bent probe) stores the last setup \mathbf{s} , probe \mathbf{p} and orientations D'_1 that are encountered during the plan extraction algorithm. If \mathbf{s} and \mathbf{p} are encountered again, then D'_1 is restored from memory, otherwise it is computed using the algorithm in Section 2.5.3.

Under the assumption that the generated plans are similar after each iteration, then this cache saves us the time to compute each D'_1 , which can be expensive. This assumption is a type of *coherence*, because we do not expect a change in a single domain to have a drastic effects on the entire plan. We will strengthen this argument with our experimental results (Section 4.5). One way to strengthen coherence is by using deterministic algorithms. We mentioned in Section 3.5.1 that randomness could be added to the clustering algorithm. Unfortunately, this has the effect of worsening the coherence assumption.

If the coherence assumption is wrong, then the planner will spend a lot of time recomputing the D'_1 cones during the plan extraction algorithm. A sophisticated controller (see previous section) should detect this problem and constrain more domains at each iteration of the main algorithm. In such, we increase the chance of extracting a valid plan, and decrease the number of times a plan is extracted. Our controller cannot handle such load balancing, but in our experience the coherence assumption holds in real-world parts.

In our implementation, the last \mathbf{s} , \mathbf{p} , and D'_1 are cached per fpoint. Therefore, they will get over-written if the fpoint is inspected more than once in the generated HLIP-tree. A solution is to store the cache per measurement, or to store a list of cached configurations at each fpoint.

A similar caching mechanism can be used to speed up plan validation. We store the valid PIPs under the assumption that memory access is cheaper than collision detection.

4.3 Points of Failure

This section covers potential failures in the planner and how they are dealt with.

4.3.1 Correct Termination

First, there is the correct termination of the planner that occurs if no solution exists, i.e., if there is (at least) a face that cannot be inspected. In this case, the user must decide if a different CMM (or a different set of probes) can do the task and run the planner again. Alternatively, the user can edit the measurement graph and inspect the face by other means. The last resort is to modify the design of the part.

4.3.2 Empty D'_1 Cone

We should never encounter a failure during hierarchical constraint satisfaction for straight probes (Section 3.5). There is the possibility, however, that we will encounter an empty D'_1 cone during the extraction of a HLIP for bent probes. This can happen at step 3 of the algorithm (Section 3.5.3). After we select a setup \mathbf{s} and probe \mathbf{p} , we extract the measurement sub-graph that corresponds to $C_{s,p}$ and assign to each leaf node a set containing a single primitive cluster that represents the allowable probe orientations, D'_1 . D'_1 can be empty for several reasons:

1. The domain stored at this leaf node has not been constrained at all, therefore any setup can be chosen. In particular, a setup from which the feature at this node is not accessible, i.e., D'_1 is empty.
2. D_1 has been computed, but the computation of D_2 has been postponed (see Section 4.1.4). The previous argument holds here too, because the setup orientation is selected from D_2 .
3. The introduction of obstacles, such as the CMM table and other fixtures, occurs after the setup is selected. These obstacles are considered during the computation of D'_1 , but not during the computation of D_2 .
4. There may be precision errors and aliasing effects due to the discrete representation of direction cones (see Section 2.5.3).

If an empty D'_1 is encountered, then we select a probe orientation from D_1 . This way we make sure that the plan extraction procedure always returns a HLIP-tree. In addition, we exclude the offending setup orientation, \mathbf{s} , from the D_2 cone under

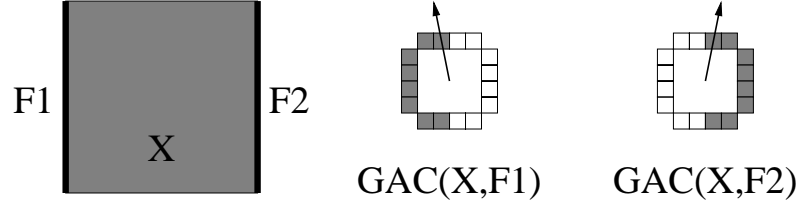


Figure 4.2: Problems with discrete direction cones

probe p . This is basically a mechanism for removing invalid PIPs from the domain of a bent probe, so that the offending setup is not encountered again.

4.3.3 Fixtures and Other Obstacles

In general, our accessibility analysis algorithms assume no obstacles besides the part. An exception is the algorithm for D'_1 for which we already have a determined setup, so fixturing elements (e.g., the CMM table) can be taken into account. We assume that for a given setup orientation we always rest the part on the table. This leads to fpoints that are approachable if the part is “floating in space”, but not when it is resting on the table.

The HLIP extractor still works correctly, but less efficiently. The verification step will detect the invalid PIPs, delete them from the appropriate domains, and another plan will be generated. By deleting the invalid PIPs we guarantee never to generate the same invalid plan.

In practice these cases are rare. To avoid them, it is wise to use setup preferences with complete fixturing information. Then, setups where the part is not resting directly on the table can be specified. Alternatively, one may limit the domains of problematic faces that can be resting on the table, e.g., by eliminating the offending setup.

4.3.4 Discrete Direction Cones

Other points of failure are in the form of sub-optimal solutions. These typically occur due to problems with the discrete representation of direction cones. A simple case is illustrated in Figure 4.2. The GACs of opposing faces of the cube are complementary hemi-spheres. The vertical (up) direction falls precisely between 2 pixels (4 pixels in

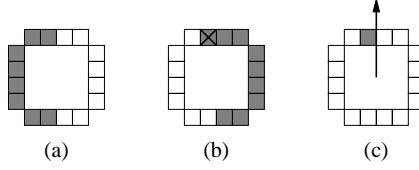


Figure 4.3: Adding preferred directions

3-D) on the top face of the cubic representation of the direction cone. This leads to 2 problems. First, directions are selected from the *center* of the pixels (as illustrated by the arrows in the figure), and therefore the vertical direction cannot be selected. Second, the intersection of the two GACs is empty, and the opposing faces cannot be inspected in a single setup (with a straight probe) or with a single probe orientation (with a bent probe).

Notice that using a finer resolution in the representation does not solve these problems. A possible solution is to jiggle the part's coordinate system slightly in the hope that the desired orientations do not fall between pixels. Another is to grow the computed cones on the topology of the cube (this can be done efficiently using digital image processing techniques [19]). The result of growing the cones is that their intersection will no longer be empty and a single setup can be chosen to inspect both faces. The problem with this approach is that it is overly optimistic and we are still left with the problem that the exact vertical direction will not be selected.

We chose a solution that solves these problems through preferred setups and orientations. The idea is to add to the computed cones the preferred directions that lie on the closure of the cone, i.e., that intersect the grown cone. These preferred orientations could be computed from knowledge about the features being inspected. For example, we know that a hole is likely to be accessible from its axial direction.

In the example in Figure 4.2, the vertical orientation falls between 2 pixels. Each direction maps to one and only one pixel. In particular, directions that fall between two pixels map to only one of them. Lets assume that they map to the pixel on their left, so the vertical orientation belongs to the GAC on the left, but not to the GAC on the right. If the vertical orientation is specified as a preferred setup orientation, it will be added to the GAC on the right. In particular, this GAC will have an additional lit pixel. Figure 4.3 (a) and (b) show the result of adding the vertical direction to the left and right GACs of Figure 4.2, respectively. The added

pixel is marked with an x . The intersection of the two cones is shown in Figure 4.3 (c). Since the vertical orientation is a preferred direction it will be selected because it maps to this pixel, even though it does not go through the center.

4.3.5 Numerical Precision

The final problem that can be encountered is *numerical precision*. In accessibility analysis we are limited by the precision of the depth-buffer. This was discussed in Section 2.4.3. The collision detection package can also encounter numerical problems. Specifically, if the probe tip is very small, then the stylus is placed very close to the point of contact. If the distance is smaller than a numerical tolerance, a collision will be detected. This problem is handled by scaling the part appropriately.

4.4 Simulation and User Interaction

The simulator is used to view the inspection plan through a graphical interface. The HLIP-tree is first linearized by a depth-first traversal of the tree. Then, the user can step through the plan by advancing to the next operator. The user can determine the granularity of the simulation by selecting the type of the operator. For example, if the user selects the **inspect-fpnt** operator, then the simulation will advance in small steps. However, if the user selects the **change-setup** operator, then the simulation will advance in large steps.

When the user iterates under the **inspect-fpnt** operator type, there is the option to view the domain associate with the fpoint. This domain is rendered as a transparent direction cone that is superimposed over the ram and probe. The GAC, D_1 , and D'_1 cones are centered about the tip and scaled by the length of the probe. The D_2 cone is centered about the origin of the ram. Figure 4.4 shows the GAC superimposed over a straight probe.

The simulator has minimal facilities for editing the knowledge base. The user can edit the domains of fpoints or completely remove undesirable fpoints from the measurement graph. In the following section we will show how these facilities are used to repair sub-optimal plans.

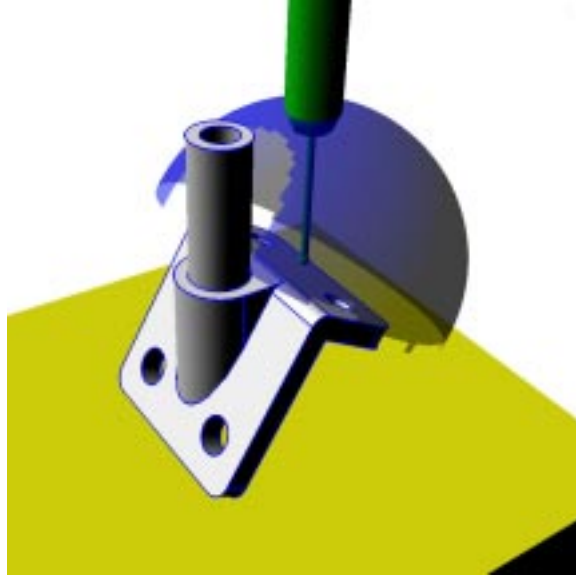


Figure 4.4: A simulation snapshot

4.5 Implementation and Results

The planner is written in C++ and executes on a Sun ULTRA 1 workstation. For collision detection we use RAPID [21], which models obstacles as “polygonal soups”. Therefore, as with accessibility analysis, we assume that the part and CMM are approximated by polyhedra. The parts are modeled with the ACIS geometric modeler [70], which provides a reliable mesh generator that is used to render the parts and provides the polygons for RAPID.

4.5.1 Toy Parts

Figures 4.5–4.10 illustrate HLIPs generated for toy parts.

Figure 4.5 has an inspection plan for the F2 part, which is a 3-D generalization of the 2-D example that was used throughout Chapter 3 (see Figure 3.1). Our planner produces the desired plan as illustrated in Figure 3.10. Notice that the top face is inspected twice — once with each probe under each tolerance. The vertical setup was specified as a preferred setup, so that the part lays flat on the table.

Figure 4.6 and Figure 4.7 illustrate the inspection of a part that is similar to F2. Here are two holes of identical diameter and depth that differ in their orientation.

Using a CMM with a fixed head (Figure 4.6), the planner computes the setups in which each hole can be inspected. Again, the top face is inspected twice — once in each setup. Using an orientable head (Figure 4.7), the part is inspected under a single probe and setup, but with two different probe orientations.

Figure 4.8 is another part that is similar to F2. This time the holes are perpendicular to each other and to the top face. They also have different diameters and depths. The result is that each is inspected by a different probe at a different orientation. Notice that the small probe cannot inspect the top face at the same orientation that it inspects the hole.

Below is the textual output of the plan illustrated in Figure 4.8. Five fpoints are sampled from each face. Notice that only 3 fpoints are inspected in the small hole, because the other two are filtered out during the knowledge base initialization step. Read “**record-meas 1 2 3 15**” as: record measurement of fpoint 15 that sits on face 3 that is part of feature 2 in tolerance 1. The indices are absolute values, e.g., feature 2 identifies a specific feature and *not* the second feature in tolerance 1. All the indices (including the probes) start from 0. The setup is denoted by the orientation (i.e., direction or unit vector) of the ram with respect to the part (in the part coordinate system). The probe orientation is denoted by a direction in the CMM coordinate system.

01) change-setup 0 0 1	22) change-orient 0.99 0.03 0.09
02) change-probe 0	23) inspect-fpnt 15
03) change-orient -0.57 -0.57 0.59	24) record-meas 1 2 3 15
04) inspect-fpnt 20	25) inspect-fpnt 16
05) record-meas 0 0 4 20	26) record-meas 1 2 3 16
06) inspect-fpnt 21	27) inspect-fpnt 17
07) record-meas 0 0 4 21	28) record-meas 1 2 3 17
08) inspect-fpnt 22	29) inspect-fpnt 18
09) record-meas 0 0 4 22	30) record-meas 1 2 3 18
10) inspect-fpnt 23	31) inspect-fpnt 19
11) record-meas 0 0 4 23	32) record-meas 1 2 3 19
12) inspect-fpnt 24	33) inspect-fpnt 20
13) record-meas 0 0 4 24	34) record-meas 1 0 4 20
14) change-orient -0.15 -0.98 -0.09	35) inspect-fpnt 21
15) inspect-fpnt 5	36) record-meas 1 0 4 21
16) record-meas 0 1 1 5	37) inspect-fpnt 22
17) inspect-fpnt 6	38) record-meas 1 0 4 22
18) record-meas 0 1 1 6	39) inspect-fpnt 23
19) inspect-fpnt 9	40) record-meas 1 0 4 23
20) record-meas 0 1 1 9	41) inspect-fpnt 24
21) change-probe 1	42) record-meas 1 0 4 24

The last two toy parts illustrate extreme cases. The first, called “swiss-block”, is a block with 10×10 holes. The second, called “swiss-sphere”, is a sphere with 20 holes. In both cases, the measurement graph contains a single tolerance for all the

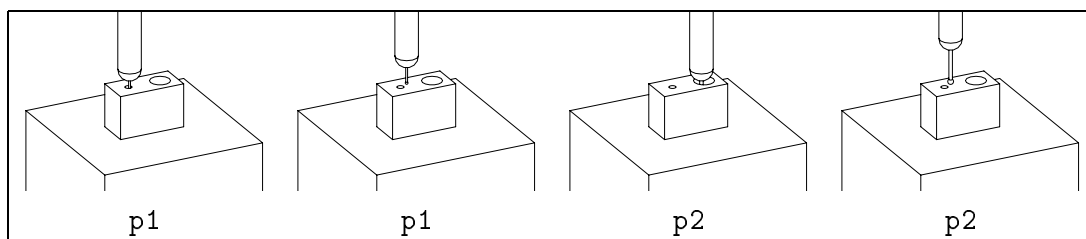


Figure 4.5: HLIP for F2 (fixed head)

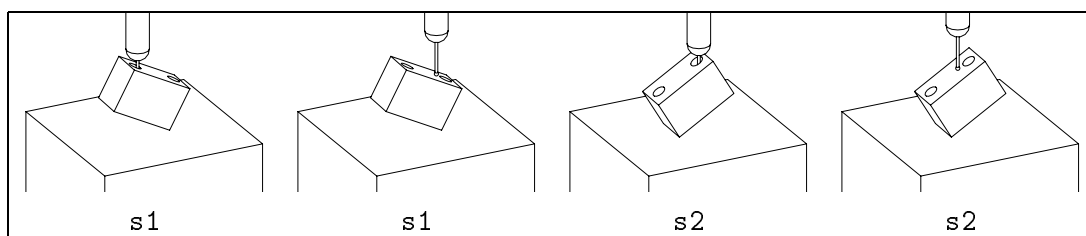


Figure 4.6: HLIP for F3 (fixed head)

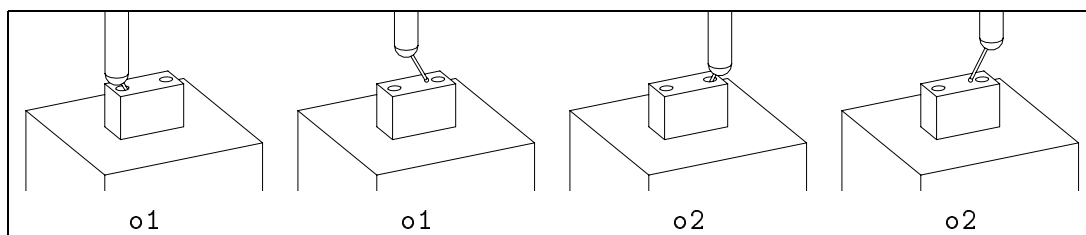


Figure 4.7: HLIP for F3 (orientable head)

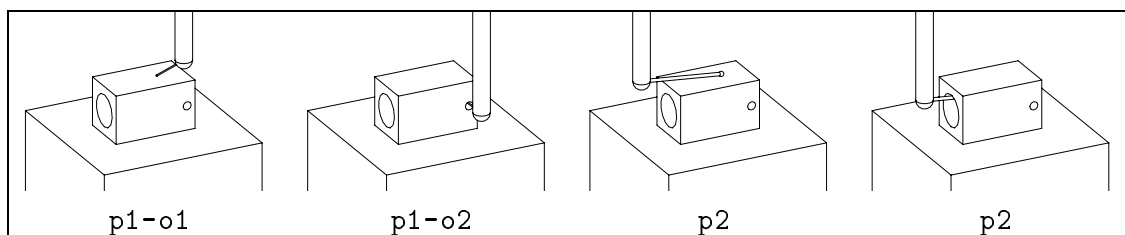


Figure 4.8: HLIP for F4 (orientable head)

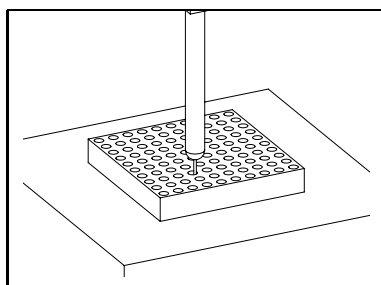


Figure 4.9: HLIP for swiss-block (fixed head)

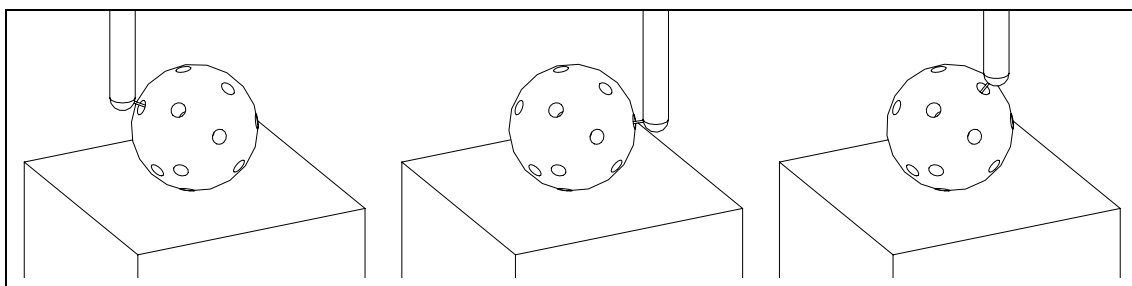


Figure 4.10: HLIP for swiss-sphere (orientable head)

holes, and the CMM has a single probe (that can fit into all the holes). As expected, the block is inspected in a single setup by a straight probe (Figure 4.9). On the other hand, the sphere cannot be inspected in a single setup by a straight probe. A different setup is needed for each hole. Using a bent probe, the sphere is inspected in two different setups — one of which is illustrated in Figure 4.10 (not all probe orientations are shown).

4.5.2 Real-World Parts

Three real-world parts, called PolySqrTa, cami2, and nclosurT, were used to test the planner. Figures 4.11–4.16 illustrate the inspection plans that were generated for these parts. All the setups and probes that appear in the HLIPs are illustrated in these figures, but only a selection of the probe orientations is shown. The HLIPs for a CMM with a fixed head are illustrated completely.

	faces	fpnts/face	nodes	edges	polys
swiss-block	106	6	4808	4812	10012
swiss-sphere	41	7	1260	974	2515
PolySqrTa	25	5	48	72	96
cam2	24	7	408	423	832
nclosurT	103	6	978	1064	1980

Table 4.1: The input parts

	probes	setups
swiss-block	1	1
swiss-sphere	1	2
PolySqrTa	3	1
cam2	2	8/0
nclosurT	2	6

Table 4.2: The number of probes and preferred setups

	tols	fters	rdtms	faces	fpnts
swiss-block	1	100	0	100	600
swiss-sphere	1	20	0	20	114
PolySqrTa	12	16	1	24	114
cam2	10	13	1	15	96
nclosurT	12	31	0	50	274

Table 4.3: The measurement graphs

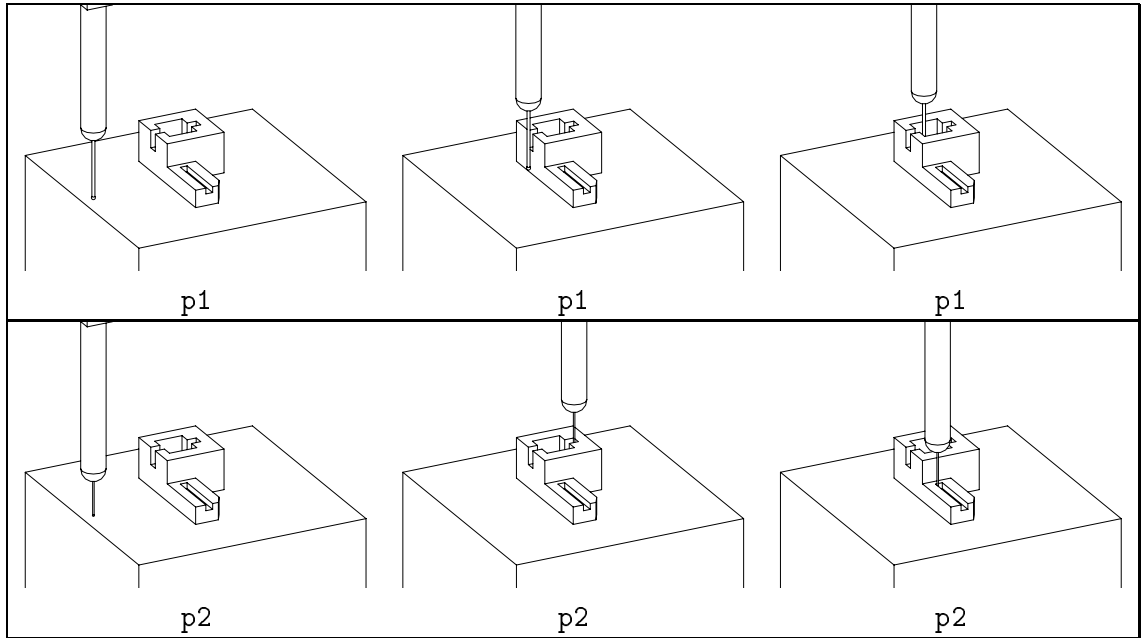


Figure 4.11: HLIP for PolySqrTa (fixed head)

Tables 4.1–4.3 show the complexity of the input models. Table 4.1 holds the size of the geometric model in terms of the number of faces and the number of fpoints that were sampled from each face. In addition it gives an idea of the complexity of the mesh that is used to render the part during accessibility analysis and the number of polygons that are used during collision detection. Table 4.2 gives the number of probes used in each experiment and the number of preferred setups that were specified. Finally, Table 4.3 shows the size of the measurement graph after the knowledge base has been initialized (some fpoints have already been filtered out at this point). Notice that the complexity of the measurement graph depends also on the arcs connecting tolerances to features and so forth, but we only supply the number of nodes in the graph. Specifically, the number of tolerances, features, datums that can rest on the table (rdtms), faces, and fpoints.

The results show that PolySqrTa can be inspected in a single setup. If the CMM has an orientable head, then a single probe (the smallest of the 3 that were supplied to the planner) can be used to inspect the entire part (Figure 4.12). Two probes are needed to inspect the part with a fixed head. A long probe is needed to reach low

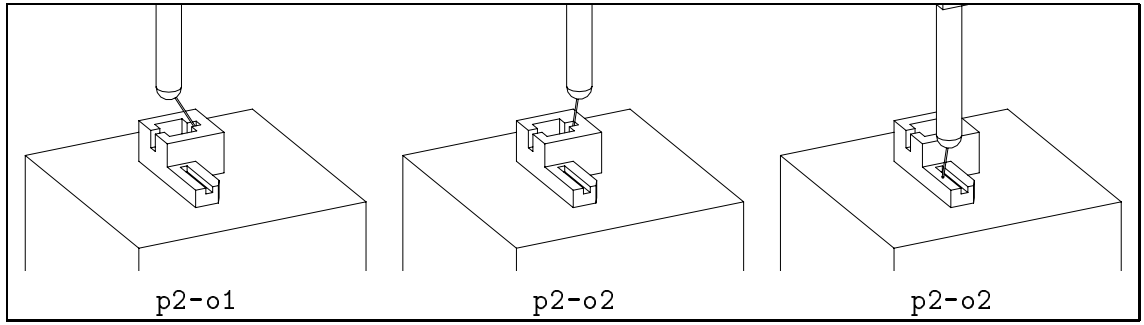


Figure 4.12: HLIP for PolySqrTa (orientable head)

points, such as the one in the top center image of Figure 4.11. Unfortunately, this probe is too thick to inspect the narrow slots, and therefore at least two probes are needed. Notice that the datum that is resting on the table is not inspected directly — we inspect the table instead. For accuracy reasons, the table is inspected with each probe, as illustrated on the left of Figure 4.11.

The cami2 part is inspected with a single probe. If the CMM has an orientable head, then the entire part can be inspected with a single setup as illustrated in Figure 4.14. This plan was generated without any explicit setup preferences. The setup was selected so as to place the primary datum on the table. Using a fixed head the CMM must inspect the part with at least 5 setups, as shown in Figure 4.13. Originally, we ran the planner without any preferred setups, but the result was that some of the setups were unstable. So we supplied 8 preferred setups — 6 for the main axes and 2 along the orientation of the large hole. The result was a plan with 6 setups — one of which was not specified by our preferences. After editing the plan in the simulator, an offending fpoint was found. This fpoint was very close to the one illustrated in the bottom center image of Figure 4.13, but could not be inspected in this orientation. Therefore, we deleted this fpoint from the measurement graph and ran the planner again. This time the desired plan was obtained, which has only 5 setups.

Figure 4.15 illustrates the HLIP generated for the nclosurT part when the CMM has a fixed head. Three setups are needed to inspect the part. Notice that the third setup is needed to inspect both features on the bottom left of the figure. Unfortunately, this is not a very stable setup. The user has three options: (1) obtain

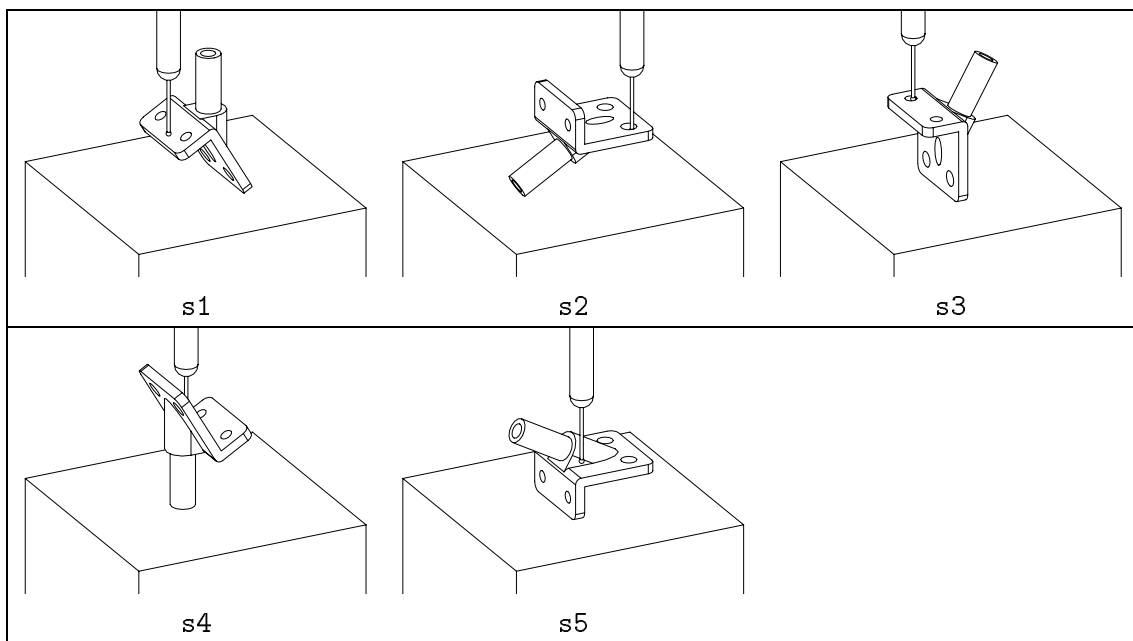


Figure 4.13: HLIP for cam2 (fixed head)

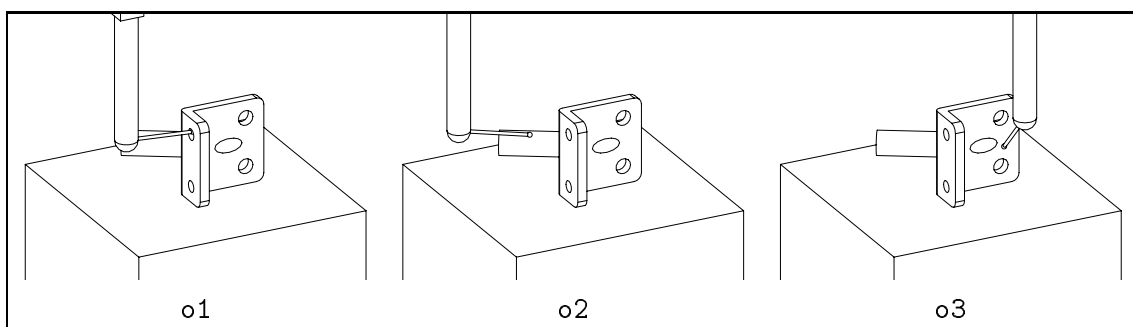


Figure 4.14: HLIP for cam2 (orientable head)

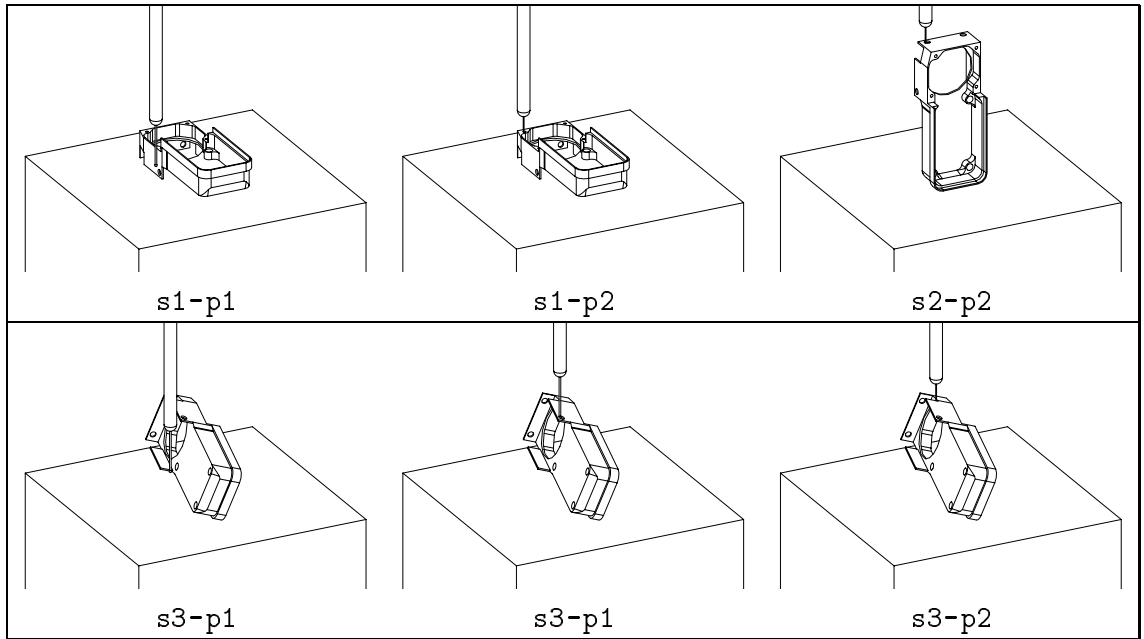


Figure 4.15: HLIP for nclosurT (fixed head)

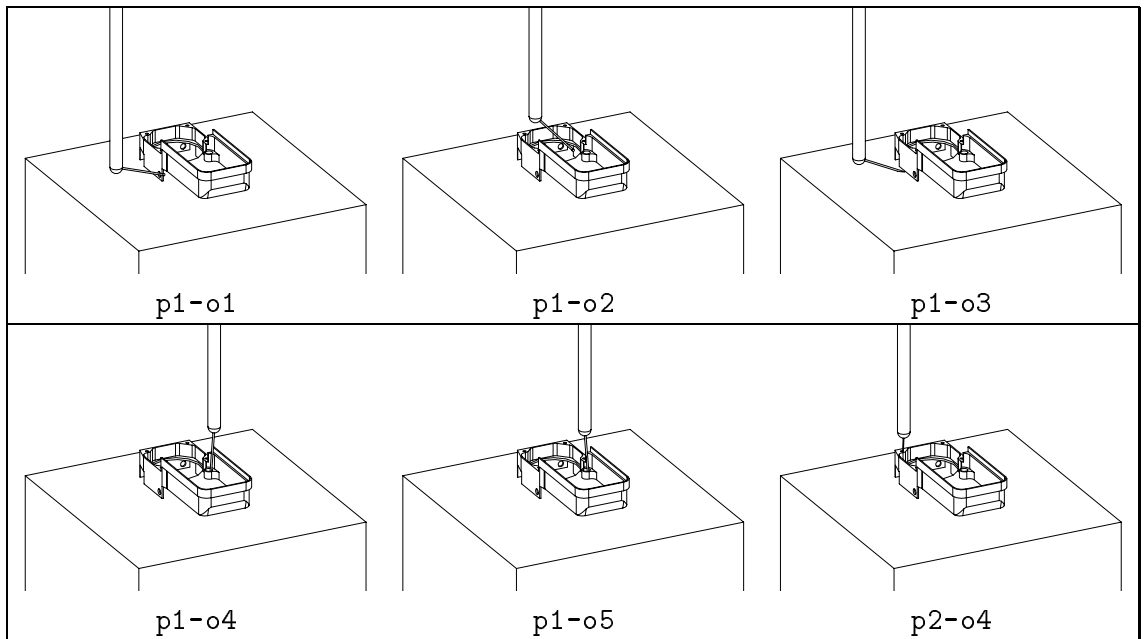


Figure 4.16: HLIP for nclosurT (orientable head)

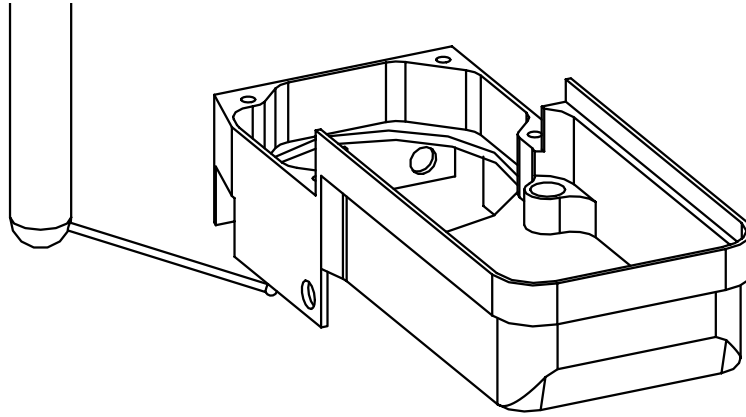


Figure 4.17: Inspecting a face that is resting on the table

a longer probe that can inspect both features in a stable setup; (2) force the planner to extract a plan with more than 3 setups, by editing the domains of measurements or by applying a weight function to the clustering process (see Section 3.6.1); and (3) use an orientable head. The latter choice produces an inspection plan with a single setup that is illustrated in Figure 4.16. This plan suffers from one abnormality that is shown in the top right image. Figure 4.17 is a magnification of this image without the CMM table. Notice that the probe is inspecting a face that is resting on the table, so the tip of the probe penetrates the table! We explain in Section 4.3.3 that these problems arise from the fact that accessibility analysis ignores all obstacles but the part itself. We chose not to test for collisions between the tip and the table, because these plans may be reasonable in some sense. For example, with the `nclosurT` part, it is reasonable to elevate the part by placing it on a block. Such fixturing information can be integrated with the preferred setups, but was not implemented with our planner.

4.5.3 Run-time Results

Table 4.4 shows the run-time results for the planner when the CMM has a fixed head. The first column shows the number of iterations the planner performed through the generate-and-test loop. The last column shows the total amount of time (in seconds) that the planner executed. The middle columns give the average

	iter	sol?	extract	valid?	KA	total
swiss-block	1	0.00	0.22	37.00	–	37.22
swiss-sphere	53	0.00	0.15	0.17	4.45	249.10
PolySqrTa	4	0.00	0.05	1.15	1.46	9.22
cam2	36	0.00	0.05	0.47	4.30	169.35
nclosurT	61	0.00	0.14	0.56	7.90	516.89

Table 4.4: Run-time results (fixed head)

	iter	sol?	extract	valid?	KA	total
swiss-sphere	98	0.00	19.70	1.15	5.12	2540.89
PolySqrTa	41	0.00	0.13	0.52	0.79	58.53
cam2	32	0.00	2.51	0.27	2.18	156.75
nclosurT	120	0.00	0.66	0.51	6.08	863.23

Table 4.5: Run-time results (orientable head)

time (in seconds) that was spent at each step of the algorithm: checking if a solution exists, extracting a plan, validating the plan, and acquiring knowledge. Table 4.5 shows the results for a CMM with an orientable head.

It is easy to see that the time spent checking if a solution exists is negligible. Also, the time spent validating the plan is amortized over the entire planning process, which shows that the coherence assumption typically holds (see Section 4.2.2). The swiss-block and PolySqrTa in Table 4.4 have long validation times on average, because the plan is produced in very few iterations. The swiss-sphere in Table 4.5 has relatively long validation time on average, because the coherence assumption does not hold. This can also be seen from the average time to extract a plan to inspect the swiss-sphere with an orientable head. At each iteration the planner generates a plan with a single setup. This setup keeps changing from one iteration to the next, hence the coherence assumption is violated. Only in a later stage does the planner compute some D_2 cones and realize that the part cannot be inspected in a single setup.

The time spent performing accessibility analysis in general outweighs the time spent performing the other steps in the algorithm. A better controller, other than finding the first invalid PIP and constraining the domain associated with it, may

	<i>GAC</i>			
	itr	prbs	sec/prb	%KA
swiss-block	0	–	–	–
swiss-sphere	52	1.00	4.45	100.00
PolySqrTa	3	3.00	0.49	100.00
cam2	33	2.00	2.28	100.00
nclosurT	60	1.87	4.23	100.00

Table 4.6: Accessibility analysis results (fixed head)

	D_1				D_2			
	itr	prbs	sec/prb	%KA	itr	prbs	sec/prb	%KA
swiss-sphere	68	1.00	2.02	27.58	18	1.00	20.00	72.42
PolySqrTa	34	2.21	0.26	61.20	5	1.40	1.12	38.80
cam2	31	2.00	1.09	100.00	0	–	–	0.00
nclosurT	89	1.81	2.07	46.17	14	1.64	15.37	53.83

Table 4.7: Accessibility analysis results (orientable head)

improve these results. We believe that the main contributions of these results are to show that the planner actually works and is very robust given very little assumptions about the input.

Finally, Tables 4.6 and 4.7 give a breakdown of the time spent during knowledge acquisition. For straight probes, we spend most of the time computing GACs. For bent probes, part of the time is spent computing D_1 cones and the other is spent computing D_2 cones. Each table shows the number of iterations spent computing each cone, the average number of probes in the constrained domain, the time (in seconds) spent computing the cones per probe in the domain, and the percentage of time spent computing these cones during knowledge acquisition. For example, on average we spend 4.23 seconds per probe computing GACs for the nclosurT part. Remember that we actually compute 4 GACS for grown half-lines (see Section 4.1.4), therefore we spend an average of 1.06 seconds computing each GAC, which is a little less than the result of Section 2.4.2.

Up to 10 directions were sampled from the D_1 cones in order to compute the D_2 cones. Table 4.7 shows clearly that it is very expensive to compute the D_2 cones.

4.6 Conclusions

We have developed a framework for generating inspection plans of high quality through the use of a constraint hierarchy. In doing so, we have decomposed the problem into knowledge acquisition and plan generation. The former is an expensive procedure that involves geometric computations, hence it is performed incrementally. By operating incrementally, it is often possible to produce a good plan without performing all the geometric computations. Plan generation is a relatively cheap procedure that involves solving the CSP using clustering techniques. The clustering approach produces satisfactory plans rapidly. Formulating and solving the problem by true optimization seems infeasible. It would require precise knowledge of the costs (in time *and* accuracy) of changing setups, probes and probe orientations, and would involve searching over continuous domains of orientations.

The generate-and-test approach is effective, because the generation step is relatively “smart”. It takes into account the approachability constraints. When these constraints are (approximately) satisfied, the generated plan is highly likely to succeed. If a plan fails the verification step, information is fed back to the generator.

The planner has been implemented and tested on real-world mechanical parts and is sufficiently fast for practical applications.

Chapter 5

Path Planning

5.1 Introduction

The focus of this chapter is on the *refinement* of a HLIP-tree into a low-level inspection plan (LLIP). A low-level plan is a *sequence* of operators, which include the high-level operators (**change-setup**, **change-probe**, etc.) and an additional low-level operator, **move-cmm-ram**. The refinement process entails *linearizing* the HLIP-tree into a suitable HLIP, and insertion of **move-cmm-ram** operators to specify a path plan for the CMM. HLIP-tree linearization involves a depth-first traversal of the tree (see Section 3.2.3). Any depth-first traversal produces HLIPs of similar quality, however the order of traversal is crucial for path planning, especially the order in which the **inspect-fpnt** sub-trees are visited. Therefore, the linearization process is not entirely independent of path planning.

In this chapter, we concentrate on the path planning problem for a part in a given setup using a specific probe in a specific orientation. That is, *we ignore the linearization of the entire HLIP-tree* and focus on the fpoints in a single sub-tree. The goal is to find an efficient and collision-free path for the CMM to inspect them all. This involves both path planning for collision avoidance, and sequencing the points to minimize the length of the path, which is a variation of the traveling salesperson problem (TSP) [13, 38, 63].

In addition, the LLIP includes a workpiece localization process after each setup is performed. The localization problem is the problem of determining the relationship between the part and CMM coordinate systems. The localization problem is treated separately from the path planning problem and is discussed in Section 5.6.

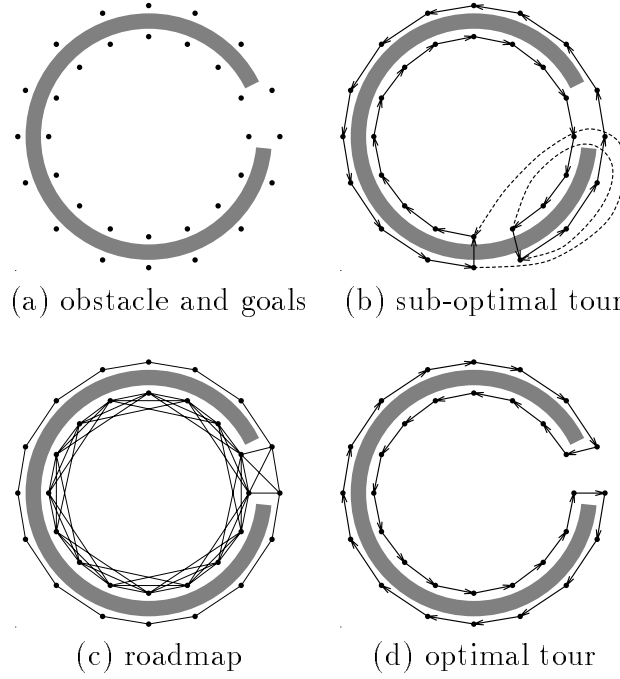


Figure 5.1: A multiple-goals path planning problem

Previous work on path planning for CMMs has concentrated on sub-problems, such as path planning between two points [44, 45, 49], sequencing the measurement points without regard to obstacles and collisions [11, 46], and path planning for a specific class of probes that are aligned with the CMM ram (i.e., straight probes) [23, 81]. Other work has focused on heuristics for CMM path planning [18, 33, 41, 86, 88] and feature-based path planning [54], which are incomplete solutions.

Sequencing the measurement points before path planning has the following drawbacks. First, it may produce sub-optimal tours, because the points are sequenced with respect to a distance function that ignores the obstacles. The second drawback is that path planning between two successive points may be very difficult. Figure 5.1a is a 2-D example of a *multiple-goals path planning problem* (defined in Section 5.3). Figure 5.1b illustrates an optimal tour of these goals without regard to the obstacle. The dashed arrows are the collision-free paths that are needed to complete the tour. Notice that in general it is not trivial to plan these paths, and that the generated plan is far from optimal.

In our scheme, we take advantage of the typically dense distribution of measurement points on the boundary of the part. Thus we expect a measurement point to have local neighbors that can be connected through simple paths. That is, a local planner, such as one that connects two neighboring points with a line-segment, will be very efficient and very likely to succeed. The idea is to create a roadmap of free-space [37] in the spirit of the probabilistic roadmap planner [31, 32], where the measurement points are nodes in the network. Once all the measurement points are in a single connected component of the roadmap, then a tour of the points is found by solving the appropriate TSP. Figure 5.1c illustrates the roadmap for the 2-D example, and Figure 5.1d is the optimal tour. Notice that we have the additional complexity of constructing the roadmap, but the local path planner used to connect nodes in the network is efficient and simple.

This chapter makes several contributions. The proposed path planner is practical and complete. In particular, the method is independent of the underlying geometry of the CMM probes and obstacles, therefore it can be easily integrated into different inspection planning systems. In fact, the method is a general framework for solving the multiple-goals path planning problem. The path planner is fast and incorporates heuristics without loss of completeness. Finally, standard TSP solvers can be used to find optimal paths in the generated roadmap.

The remainder of the chapter is organized as follows. First, we discuss related work. Next, we describe a general method for solving the multiple-goals path planning problem. Then we describe the heuristics that are used to drive the CMM path planner. Finally, we present our implementation and experimental results. Before concluding this chapter, we briefly discuss the localization problem and how it relates to CMMs.

5.2 Related Work

5.2.1 General Path Planning

Robot path planning has been studied intensively over the past two decades [24, 37]. The classic path planning problem is to find a collision free path for a robot to get from an initial state to a goal state within a world of static obstacles. There are

many variations of the problem, such as obstacles in motion or finding an optimal path for the robot.

The *configuration space* (C-space) is a mathematical representation for the path planning problem [47, 48]. All possible configurations of the robot are mapped to points in C-space. The *free-space* is the subset of C-space that contains all the valid configurations of the robot, i.e., the configurations in which the robot does not collide with any obstacles. Each obstacle is mapped to a subset of C-space, called a *C-space obstacle*, which includes all the configurations in which the robot collides with the obstacle.

The following is a categorization of different approaches to path planning. Each is a general method for solving the problem that may be practical for a specific domain. Much depends on the specific problem, the resources available and the expectations from the resulting path.

- **Roadmaps** are graphs that are embedded in free-space — nodes are valid configurations and arcs are collision-free paths. Finding a solution path involves connecting the robot’s initial and goal configurations to the graph and then searching the graph for a path between these connections. A variety of roadmap techniques have been developed including visibility-graphs [48], retraction graphs (i.e., skeletal maps, such as Voronoi diagrams), freeway graphs [6], and silhouette methods [9]. The freeway method is a heuristic technique that works well in low dimensions with scattered environments (e.g., for mobile robots). The other roadmaps are expensive to compute. Our work is derived from the probabilistic roadmap planner [31, 32] that will be described in Section 5.3.
- **Cell decomposition** of free-space is another graph embedding. This time the nodes are the cells, with adjacent cells (implicitly) connected by an arc. A path from the robot’s initial configuration to the goal configuration is produced from a *channel* of cells connecting the two configurations [37].
- **Potential field methods** use a potential function over free-space to guide the robot from the initial configuration to the goal configuration [37]. Obstacles repel the robot, while the goal configuration attracts it. The robot typically

follows the gradient of the potential function to get closer to the goal. Local potential functions are largely affected by local obstacles, hence these planners often get stuck in local minimums and are incomplete. However they tend to be very fast and there are techniques to escape local minimums (e.g., simulated annealing or random walk).

5.2.2 CMM Path Planning

Walker and Wallis describe an inspection planning system that performs path planning for a CMM with a probe that is aligned with the ram axis (i.e., a straight probe) [81]. Generating a path between the probing points involves ordering the points and then connecting them. They observe that the ordering of the points is a variation of TSP, and propose a greedy method that orders the points hierarchically: first by the feature that they belong to and then by choosing the next point that is closest using the natural distance metric. Generating a path between two probing points is done via a straight line segment. If this path collides with the part then two mid points are added: one above the current point and the other above the next point, both at a height that is safely above the part. This path is feasible, because the probe is abstracted by a cylinder, which gives it the property that it can safely approach points along its axis. They suggest improving this scheme by a heuristic that traverses the boundary of the part to find a shorter path, which resembles a greedy visibility-graph method [37, 47, 48].

Gu and Chan report on another system that performs path planning for straight probes [23]. Their method is very similar to Walker and Wallis' method, except in the manner that paths are tested for feasibility. Gu and Chan compute the volume swept by the probe and check for interference with the part, while Walker and Wallis compute the C-space obstacle and perform line classification. It was cheaper for Walker and Wallis to compute the C-space obstacle, because they used a CSG representation [65], which implies that null-object detection is expensive, but line classification is cheap. Walker and Wallis do not detail in their paper how they grow the CSG model, which is a non-trivial problem [67]. From the figures in their paper [81], they do not grow the model correctly.

Khoshnevis and Yeh describe a CMM inspection planner [33]. Unlike the systems above, they assume that the probe is orientable, i.e., it is not necessarily aligned with the ram axis. They contribute two ideas: first they reduce the 3-D path planning problem into a 2-D one, then they use 2D accessibility analysis to generate path plans. The reduction is performed by slicing the part model with planes that are parallel to the ram and planning a path within each slice. The paths are then connected using slices from planes that are perpendicular to the first ones. The main problem with this scheme is that the planner is incomplete and largely depends on the orientation of the slicing planes.

Yau and Menq describe path planning for dimensional inspection [86, 88]. They perform path planning in a hierarchical manner: find a path for the tip, verify the path for the stylus and then for the ram. The initial path is a line segment connecting the two probing points. If collisions are detected, then correction routines are called based on heuristic rules to modify the path. The rules work in common situations, but otherwise the planner is incomplete. Lim and Menq [41] expand on this work by performing global accessibility analysis before path planning (Yau and Menq just perform local accessibility analysis). This accessibility information is used to cluster probe directions and perform path planning on single directions. The initial default path between two probing points is modified by adding approach paths to the probing points. These paths are in the direction of the probe orientation, hence minimizing the volume swept by the probe.

Some effort has been applied to find an optimal path for a CMM between two measurement points [49, 44, 45]. The general problem of finding the optimal path between two configurations has been solved using the visibility-graph algorithm [37, 47, 48]. Lu, Ni and Wu propose an octree representation for the part in order to simplify the computation of the C-space obstacle [49]. Limaïem and ElMaraghy use a cell decomposition of C-space and then Dijkstra’s shortest path algorithm to find the optimal path [44]. Lin and Murugappan abstract the CMM probe as a point and use ray shooting techniques to find the path [45]. We feel that finding the optimal path between two arbitrary points has been solved in the general case and is not critical for CMM path planning, which needs to address multiple points. Most measurement points have close neighbors, therefore the path between them is typically trivial and can be resolved by a local planner.

Cho and Kim concentrate on measurement point selection and sequencing for inspection of sculptured surfaces [11]. In particular, they investigate results of sequencing the points using several greedy algorithms and subdivision algorithms for approximating the TSP. Collision avoidance is not the focus of their work. Similar work on measurement point sequencing is documented by Lin and Chen [46]. Merat and Radack exploit a feature-based system to predefine measurement points and paths as part of the feature definition [54]. The system lacks the capability to automatically repair plans that are infeasible due to interacting features.

5.3 General Method

The CMM path planning problem can be generalized as a *multiple-goals path planning problem*, which is an instance of a shortest path problem [57]. Given a *robot*, a set of *obstacles*, and a set of *goal configurations*, the problem is to find a shortest *tour* of the goal configurations, or notify that no such tour exists. We define a tour to be a closed and collision-free path that visits each goal at least once. Optimality is not clearly defined in the general case [57], but for CMMs we wish to minimize the length of the path.

The multiple-goals path planning problem is the join of two classic problems: path planning and network optimization. Unlike standard network optimization problems, however, the structure of the given graph is not explicitly given, but is implied in the structure of the underlying C-space. In particular, it is not known a-priori if a path *exists* between two configurations, let alone what the shortest path *is*.

It is easy to see that a solution to the multiple-goals path planning problem exists iff all the goal configurations are in a single (path) connected component of free-space. The optimal solution, if one exists, is a tour of the goal configurations, where the path between successive goals is the shortest (otherwise we can obtain a shorter tour by replacing this path with the shorter one). This simple observation leads to a naive algorithm that solves the multiple-goals path planning problem. First, construct a network where the nodes are the goal configurations, and each pair of nodes is connected by an edge that represents an optimal path. Then, extract an optimal tour by solving the TSP. Unfortunately, this is not a very practical

approach, because not only is finding an optimal path between two configurations NP-hard [37], but TSP is NP-hard as well [13, 38, 63].

In this section, we present a *practical* path planner. The planner is practical in the sense that it will most often find a solution (if one exists) in a reasonable amount time, and this solution will be near optimal. The planner is *probabilistically complete*, i.e., it will find a solution if left to run long enough. If the computational resources are available, then the planner can incrementally return plans that converge to an optimal solution. The planner can be implemented once for a variety of robots, and is flexible enough to allow the integration of domain knowledge that is specific to each robot.

Our proposed planner is based on the probabilistic roadmap planner (PRM) [31, 32] and works in two steps: (1) construct a roadmap that includes the goal configurations in a single connected component (or decide that they cannot be placed in a single component, hence no solution exists), and (2) extract an optimal tour from the roadmap (here optimality is with respect to the roadmap).

5.3.1 Roadmap Construction

The *roadmap* is an undirected graph $G(V, E)$ that captures the connectivity of free-space [37]. The nodes, V , correspond to configurations in free-space, while the edges, E , are collision-free paths. Each edge is assigned a *weight* that is equivalent to the length of its path.

The elementary operation in our planner is the incremental insertion of nodes into the roadmap. This is identical to the procedure used in PRM and is as follows:

```

PROCEDURE Insert( Node v, Roadmap G(V,E)
if v is not in free-space then:
    return failure.
otherwise:
    let U be the neighborhood of v in V.
    add node v to V.
    for every u in U do:
        call local planner to find a path between u and v.
        if a path is found then:

```

```

let L be the length of the path.
add edge {u,v} to E with weight L.

```

The *neighborhood* of a node is defined as the nodes that are nearest to it in the roadmap. The *local planner* should be an efficient path planner, but not necessarily a very powerful one (i.e., it can fail to find a path even if one exists). The planner should most often succeed in finding paths between nodes that are relatively close to each other, i.e., neighboring nodes. A simple local planner is one that tests paths that are line-segments in the configuration space [31, 32].

Finally, we construct the roadmap in the following manner:

1. **Initialization Step:** Create an empty roadmap and insert the goal configurations.
2. **Enhancement Step:** Incrementally insert random nodes (or nodes selected with domain specific knowledge) into the roadmap until all the goal configurations are in a single connected component of the roadmap (or determine that they cannot be all connected, hence no solution exists).

The algorithm for constructing the roadmap is identical to its PRM counterpart [31, 32], except that we make sure to include the goal configurations. Using similar arguments as with PRM, it is clear that the planner will capture the connectivity of free-space if the algorithm is run indefinitely, hence it is *probabilistically complete*.

Domain knowledge is incorporated via the local planner and the enhancement step. These are heuristics that can greatly speedup the roadmap construction. We cover CMM heuristics in Section 5.4.

5.3.2 Optimal Tour Extraction

The constructed roadmap has the goal configurations all in one connected component, hence it is clear that there exists a path (in the roadmap) that traverses all of them. This is not a *tour* of the roadmap, because the roadmap contains superfluous nodes that were added during the enhancement step — these nodes do not need to be traversed by the path. In addition, the path is optimal with respect to the roadmap, hence it may not be the optimal path in reality. If the path is far from

optimal, then the roadmap can be incrementally refined (by the enhancement step) to better capture the structure of the free-space. However, since TSP is NP-hard, its solution is approximated anyway.

A shortest path is found in a two step process:

1. **Preprocessing Step:** This is an optional step that involves preprocessing the roadmap to meet the requirements of the specific TSP algorithm in use.
2. **Solve TSP:** Solve the problem by applying a TSP algorithm to the roadmap.

The optional preprocessing step augments the roadmap into one that is more suitable for classic TSP solvers. Classic solvers typically assume the input in the form of a distance matrix, i.e., a complete directed graph with weights assigned to each edge. Our roadmap is inappropriate for the following reasons: (1) it contains superfluous nodes, (2) it is not a complete graph, and (3) it does not fulfill the triangle inequality. The triangle inequality is highly desirable, because then there are TSP algorithms that will find an approximate solution within a constant ratio from the optimal one [13, 38, 63].

Fortunately, the roadmap may be augmented in polynomial time. We create a new roadmap that contains only the goal configurations. Then for every pair of nodes in the new graph, we create an edge that is the shortest path in the original roadmap. The *all-pairs shortest-path* problem is of polynomial complexity [13]. Note, that if memory requirements are tight, then just the length of the shortest path is needed, because the path itself can be extracted from the original roadmap. (If the local planner is deterministic, then the paths need not be stored in the roadmap either [31, 32].)

5.3.3 Example

Figure 5.2 illustrates the planning algorithm on a simple 2-D example. We assume that the neighborhood of a point consists of those points that are within a predetermined distance. The local planner that is used in this example connects neighboring points with line-segments. If the line-segment is a collision-free path, then the local planner returns it, otherwise it fails. The initialization step produces the roadmap in Figure 5.2a. Free-configurations are added randomly until the goal

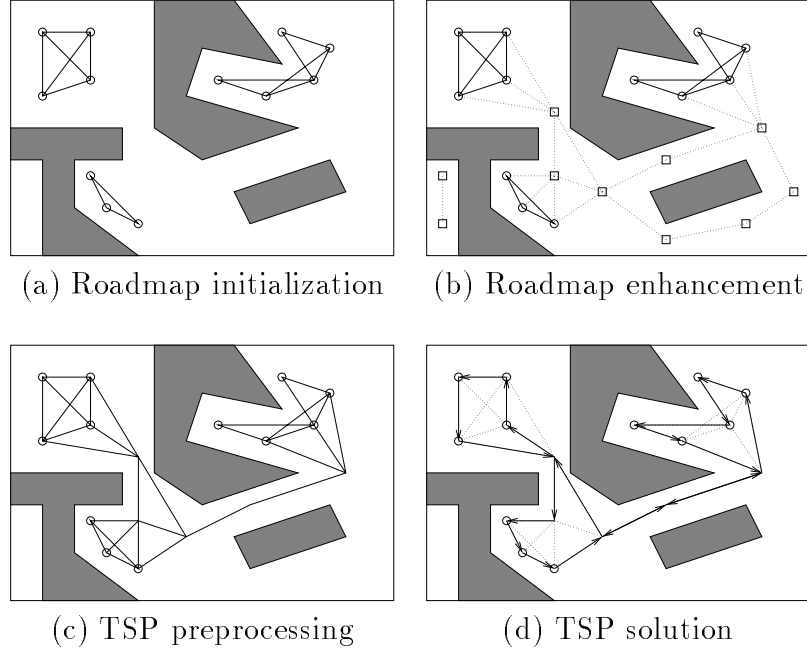


Figure 5.2: A 2-D illustration of the path planning algorithm

configurations are in a single connected component as in Figure 5.2b. Goal configurations are marked as circles and random nodes are marked as squares. Notice that the roadmap as a whole is not connected, because of the disconnected component in the bottom left corner. The TSP preprocessing step removes the nodes that were added in the enhancement step and produces the roadmap illustrated in Figure 5.2c. Notice that the new edges may be more complex than line-segments, and some of the paths overlap. The result is a connected roadmap on which we apply a standard TSP algorithm to produce the path illustrated in Figure 5.2d.

5.4 CMM Heuristics

In this section we cover the various elements of the multiple-goals path planning problem with respect to CMMs. We introduce domain specific heuristics that guide the local path planner and the enhancement step in the general algorithm.

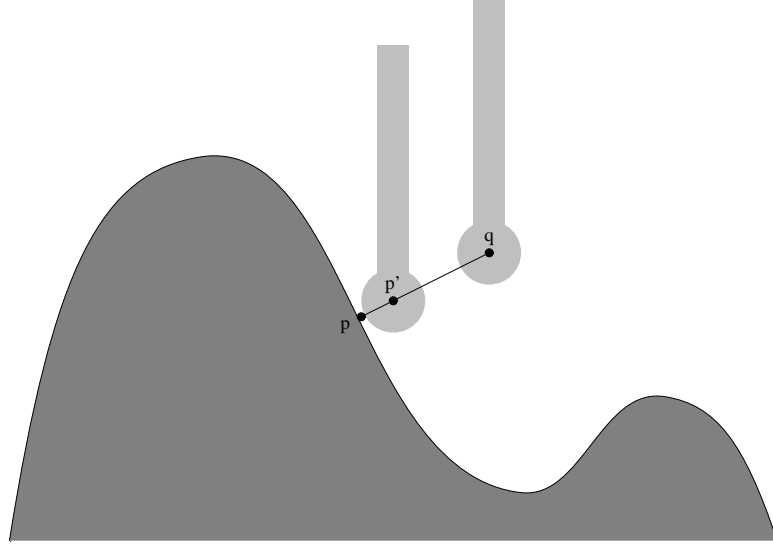


Figure 5.3: The approach/retract path

5.4.1 The Domain

The robot is a CMM. This is a holonomic robot where the ram has 3-dof for translations in the X , Y , and Z directions. Velocity and other motion constraints are insignificant in our problem. We assume that a specific probe is fixed to the ram in a predetermined orientation, therefore the C-space is 3-D.

We also assume that the probe is a touch-trigger probe that contains a single spherical tip. The idea is to limit the number of positions from which a probe can measure a point. If the point is a non-singular point on a surface, then a spherical tip is in contact with this point in at most one position. A probe with multiple tips (e.g., a star probe [5]) often can contact a point with more than one tip. This adds another level of complexity to the problem, and is not discussed here.

The obstacles in our domain are the part that is being inspected, the CMM table and the fixturing elements. The position of the part on the table is determined by the part setup.

The input to the path planner is a list of fpoints, which are the children of a probe orientation node in the HLIP-tree. For each fpoint, the setup, probe and probe orientation are known, and therefore there is a unique pose in which the specified tip's center coincides with the point.

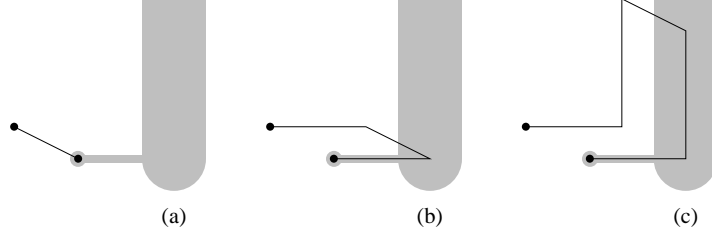


Figure 5.4: Plans generated by the local planner

Recall that if p is the point to be measured on the surface of the workpiece, then placing the center of the tip at p will cause the probe to penetrate the part (see Figure 5.3). Instead, we place it at the offset point $p' = p + r\vec{n}$, where r is the radius of the tip and \vec{n} is the normal to the surface at p . (We assume that p is not singular, because it is not wise to measure a singular point with a tactile probe.)

It is common to have an approach/retract path for every measurement point [11, 88]. The idea is to minimize errors and possible crashes, when the probe approaches a measurement point. For maximum accuracy, the approach direction should be normal to the surface at this point. To minimize crashes and increase accuracy, the approach is done at low velocity. (Remember that the manufactured part is never perfect.) The high-level planner verifies the approach/retract path, $q \rightarrow p' \rightarrow q$, for each point p that is to be inspected (see Figure 5.3), therefore this is not repeated by the path planner. Instead, we plan a path between the retraction points, q , and trivially insert the approach/retract path into the final plan.

5.4.2 Local Planner

The local planner is an *efficient* path planner, but not necessarily *complete*. We expect it to most often succeed in finding paths between nodes that are relatively close to each other. We use a generate-and-test planning paradigm — generate a path between two nodes and test for collisions. If the planner fails to find a simple path, then more complex ones are tried before the planner returns failure.

The geometry of the CMM ram and probe suggests simple paths that are likely to be collision-free. They are listed in order of complexity:

1. Translate along the line-segment that connects the two nodes (Figure 5.4a).

2. Translate along the axis of the probe, then translate along a line-segment parallel to the one above, and finally translate back along the axis of the probe (Figure 5.4b).
3. Translate along the axis of the probe, then translate along the axis of the ram. Next, translate along a line-segment that is parallel to the first. Finally, translate back along the ram axis, and then back along the axis of the probe (Figure 5.4c).

The distance of translation along the axis of the probe or ram depends on the position of the obstacles. Simple collision checks, such as the heuristics suggested by Yau and Menq [86, 88], can be used to generate these paths. Notice that for straight probes the last two paths coincide and are similar to those generated in [23, 81].

If two neighboring fpoints share a common face and the planner fails to find a simple collision-free path between them, then 2-D path planning can be employed. Assuming that the entire face is accessible to the CMM in the given configuration and the face is connected, then a collision-free path exists between the two points along the surface of the face. Such a planner is hard to implement and was not attempted by us. We feel that the whole principle of the roadmap method is to keep the local planner simple and efficient. (An alternative to 2-D path planning is suggested by the enhancement step in the following section.)

We described heuristics for *generating* simple paths for the CMM. *Testing* these paths for collisions can be performed in object space or C-space. Computing the C-space is an expensive procedure, so we chose to work in object space. Computing the sweep of the ram and probe abstractions is trivial, when they are abstracted by cylinders and spheres. Furthermore, the exact boundary of the sweep is not needed for collision detection — we need a superset of the boundary that is contained in the sweep. Therefore, each component of the ram and probe is swept independently.

Figure 5.5 illustrates the result of sweeping a ram and probe along simple paths: (a) stationary, no path, (b) line segment between two neighboring points, (c) approach/retract path, and (d) translation along the axis of the probe. The snapshots for this figure were taken from the inspection plan simulator.

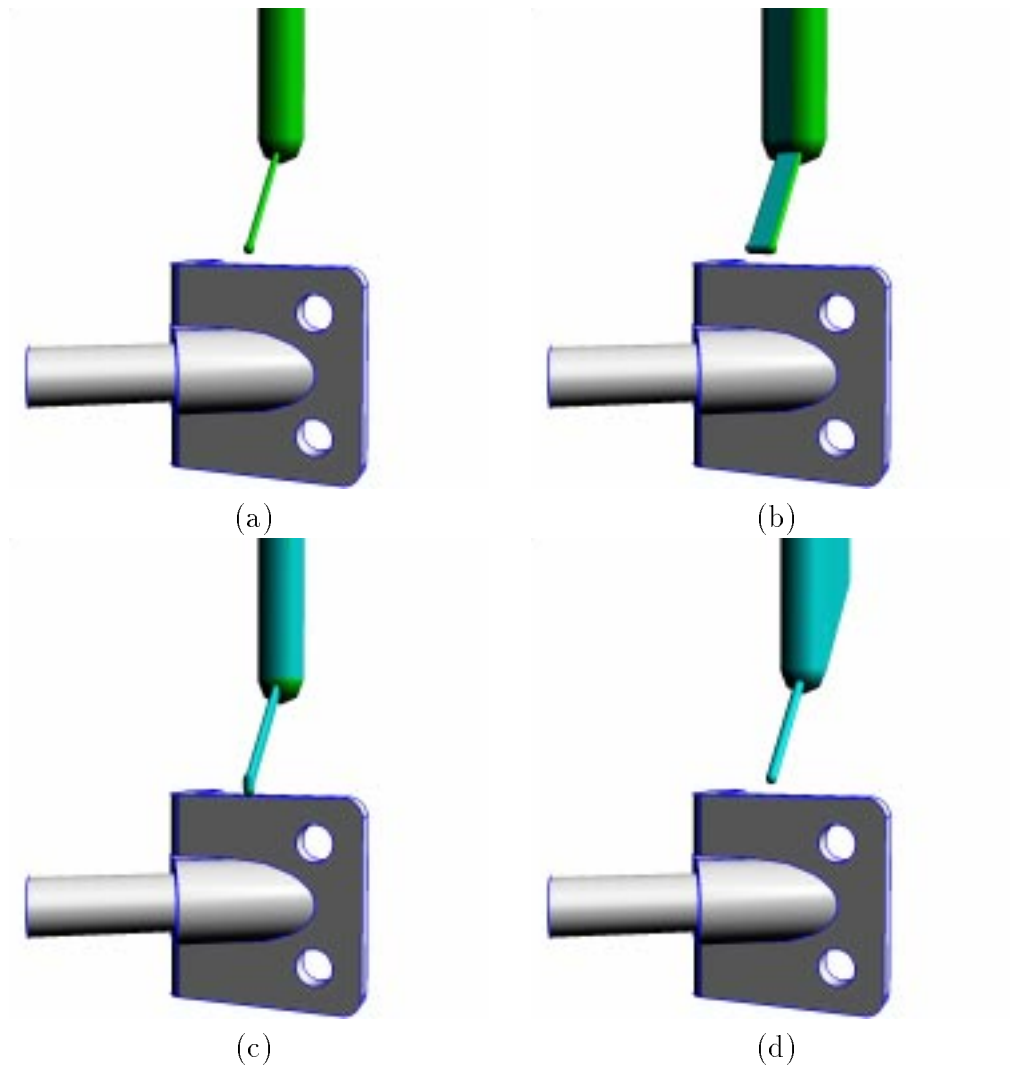


Figure 5.5: The swept ram and probe

5.4.3 Enhancement Step

The roadmap is constructed in two phases. During the *initialization step* the goal configurations (i.e., measurement points) are inserted into the roadmap. If all the measurement points are in a single connected component of the roadmap, then a tour of the points exists. Otherwise, we enter the *enhancement step* in which additional nodes are incrementally inserted into the roadmap in the hope of merging the disconnected components.

Again, several strategies for selecting additional nodes are available. We list them below:

1. If two fpoints on a common face are disconnected in the roadmap, then select additional points along the surface of the face. The idea is that the face is typically accessible and connected, so we expect a *dense* sample of points on the face to be in a single component of the roadmap.
2. Select points that are translations of the fpoints along the axis of the probe and/or ram. These points have a high likelihood of being in free-space and further away from the obstacles. Therefore, we expect it to be easier to connect such points through a local planner.
3. Select points at random in the spirit of PRM.

Notice that the first two strategies are CMM heuristics, whereas the last one is a general technique. It is easy to see a similarity between the heuristics introduced here and the ones used with the local planner. There is a tradeoff between the local planner and the enhancement step — a powerful local planner is typically less efficient, but then less enhancement steps are needed. Since 2-D path planning is hard to implement and would degrade the performance of the local planner, it makes sense to sample points from faces as suggested above.

5.5 Implementation and Results

The planner has been implemented in C++ on a Sun ULTRA 1 with Creator 3D graphics hardware, Solaris 2.5 and 128 MB of memory. We use RAPID for collision

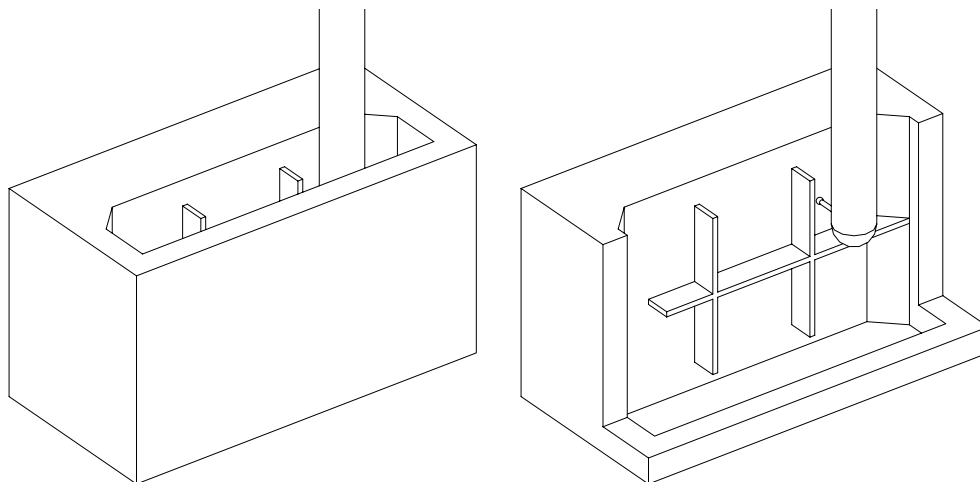


Figure 5.6: An example part (left) and a section (right)

detection [21]. RAPID models obstacles as polygonal soups, therefore we assume that the part and CMM are approximated by polyhedra. For conservative approximations the ram and probe are grown slightly. This also ensures that the probe does not get too close to the part during the inspection process (remember that the approach/retract path is verified independently, so the probe should *not* come in contact with the part during path planning).

The path planner successfully generated plans for the real-world parts that were introduced in Section 4.5.2. It would be futile to illustrate these plans in the text. Instead, we use a toy part that can be projected conveniently into 2-D. Figure 5.6 shows the part. The section on the right reveals a planar face with an H-shaped barrier. The CMM cannot reach the bottom of the face directly, except through a narrow opening on the left hand side. This is a non-trivial 2-D path planning problem that is embedded in 3-D.

Figures 5.7 and 5.8 show the initial roadmap and path plan for 30 and 100 goals, respectively. Some of the criss-cross in the path plan is due to the fact that the robot must backtrack from the dead-end in the bottom-right of the H-shaped obstacle. Notice that the initial roadmap with 30 goals is disconnected, which is remedied through the enhancement step. We see that in some sense the path planning problem is easier (i.e., the chance of failure is smaller) the denser the goals are. Table 5.1

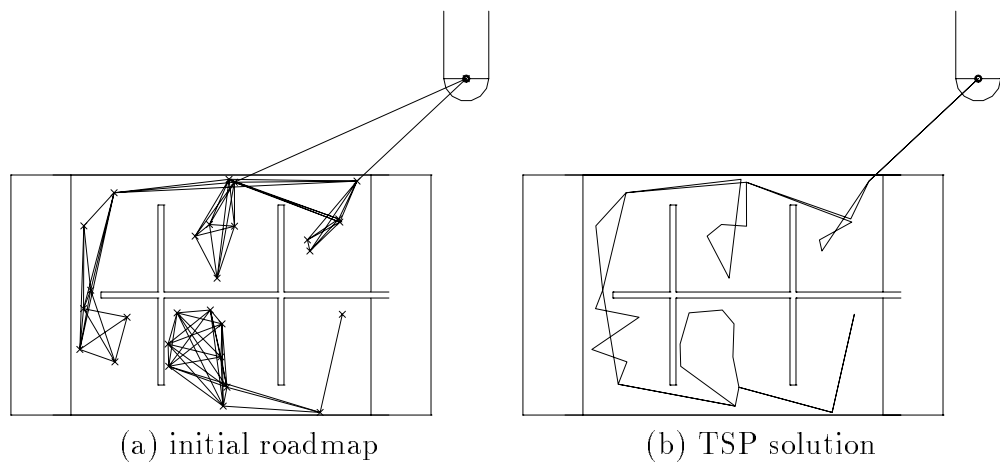


Figure 5.7: Path planning with 30 measurement points

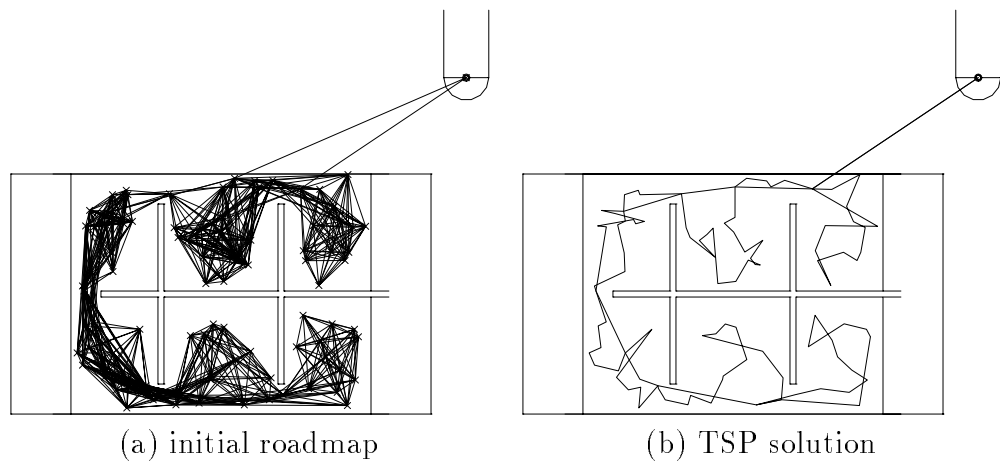


Figure 5.8: Path planning with 100 measurement points

initialization		enhancement		TSP	total
# goals	sec	# added	sec	sec	sec
30	5.13	73	7.33	0.50	12.96
100	22.15	0	0.00	2.33	24.48

Table 5.1: Path planning run-time results

shows the run-time results of these two experiments. During the enhancement step of the roadmap with 30 goals, 73 nodes were added at random before the measurement points were in a single connected component. The large number is due to the fact that the random nodes need to fall on very narrow areas of free-space. We did not implement the selection of points along the surface of the face, which would have greatly reduced the number of added nodes.

Solving the TSP was not the focus of this research. The problem has been well studied and powerful algorithms are available [38, 63]. We preprocess the roadmap using an *all-pairs shortest-path* algorithm and then solve the TSP using the *closest-point heuristic* [13]. The closest-point heuristic guarantees that the obtained tour is no longer than twice the optimal. In practice, better algorithms are available. Also, since inspection planning is performed off-line, then a lot of resources can be spent to obtain near optimal (or even optimal) solutions.

5.6 The Localization Problem

The multiple-goals path planning problem assumes a perfectly known workspace environment. In particular, we assume to know the relationship between the part and CMM coordinate systems. Without knowing this relationship, a measured point would be different from the target one, and there would be a high risk of probe crashes. Relating the part and CMM coordinate systems is known as *the workpiece localization problem* [22, 40, 53]. This problem occurs every time the part is setup on the CMM table, unless care is taken to fixture the part in a known position and orientation.

Traditionally, datums are inspected first to establish a reference frame for the part [23, 53]. This is known as the 3-2-1 approach, and is typically limited to planar

datums that are orthogonal to each other. First, 3 points are measured from the first datum to establish a plane, then 2 points are measured from the second datum to establish a second plane that is perpendicular to the first, finally a single point is measured from the last datum that is perpendicular to the first two.

Modern techniques solve the localization problem for parts with arbitrary surfaces [22, 40, 53]. The idea is to measure points on the part and then fit them to the model using iterative techniques, such as least-squares.

All the methods above use an initial set of measurements to solve the localization problem. However, they do not address the problem of obtaining the initial measurements. This is the familiar chicken-and-egg problem. On the one hand, we cannot use the CMM to measure the points without first establishing the location of the part. On the other hand, we cannot establish the location of the part without these measurements. To solve this problem we assume one of the following:

- The part is positioned with care on the CMM, so that the desired setup is near perfect. This can be accomplished with dedicated fixtures and other tools. Then the CMM can perform measurements with a high confidence on the location of the part.
- The initial measurements are retrieved manually, e.g., by physically guiding the CMM or using a hand-held stylus [51]. Then the relationship between the part and CMM coordinate system can be determined using one of the above techniques.
- Other sensors are used to determine the location of the part in a safe manner. Particularly useful are non-contact sensors, such as cameras and laser range finders.

If the part has pronounced planar datums, then it makes sense to measure these surfaces, e.g., using the 3-2-1 approach. In this case, the path planner should order the points on the datums before the points on the rest of the part. That is, the TSP problem should be solved separately for the points on the datums and for the rest of the points in the roadmap.

Notice that with the iterative techniques, the more measurement points the more accurately the localization problem is solved [22, 40, 53]. Due to this fact, it makes

sense to incrementally solve the problem as more points are measured. This observation is useful to the program that interprets the measured data and drives the CMM, but is not part of the planning problem.

5.7 Conclusions

In this chapter we presented a general framework for solving the multiple-goals path planning problem. The method extends the probabilistic roadmap planner to include the multiple-goals as part of the network, and then extracts an optimal tour (of the roadmap) by solving the underlying traveling salesperson problem. The planner has been implemented successfully for CMM path planning, which is a non-trivial 3-D path planning domain for real-world applications.

The following items summarize the characteristics of the proposed approach that make it suitable for *practical* applications:

1. **General Framework:** The planner provides a mechanism of solving the multiple-goals path planning problem for *arbitrary* robots. Therefore, the main algorithm can be implemented once for a variety of domains.
2. **Domain Heuristics:** Domain specific knowledge can be integrated into the planner (through the local planner and roadmap enhancement step) without loss of probabilistic completeness.
3. **Collision Detection:** The configuration space does not need to be computed explicitly in order to construct the roadmap.
4. **Incremental Construction:** The roadmap may be enhanced incrementally to obtain better solutions.
5. **TSP Solvers:** Standard TSP algorithms can be used to find near optimal tours in the constructed roadmap.

6. **Visualization:** The roadmap can be visualized by a user (at least when the configuration space is of low dimension), which can aid in analyzing the planning problem. For example, if the planner fails to connect the goal configurations in a reasonable amount of time, then a user may view the roadmap and decide if it can be connected, or that no solution exists.

Chapter 6

Conclusions

6.1 Summary

This dissertation focuses on dimensional inspection planning for CMMs. Given a solid model of an object, including dimensioning and tolerancing information, and a model of a CMM, the goal of inspection planning is to generate a program. The program should drive the CMM through the inspection of a manufactured part. The inspection process should be efficient and provide enough data to determine if the part conforms to the solid model or not.

Initially, our planner generates a high-level inspection plan (HLIP), which specifies how to setup the part on the CMM table, which probes to use and how to orient them, and which measurements to perform with each setup, probe and probe orientation. The HLIP is then expanded into a complete program for driving the CMM to inspect the object. Chapters 2–4 discuss the high-level planner from the basic tool of accessibility analysis to the overall system architecture. Chapter 5 is dedicated to CMM path planning, which is used to refine the HLIP into a low-level inspection plan (LLIP). The LLIP includes a complete path plan for the CMM.

Accessibility analysis is a spatial reasoning activity that seeks to determine the directions along which a tool or probe can contact a given portion of a solid object's surface. These sets of directions are called direction cones. We provide a suite of algorithms to compute global accessibility cones for a variety of probe abstractions.

The high-level inspection planning problem is mapped into a constraint satisfaction problem (CSP), where the variables are the measurements to be performed. Each variable has a domain of allowable values that is represented by a collection of

direction cones and additional information, such as applicable probes. Accessibility analysis is used to constrain the domains appropriately.

Hierarchical constraints are defined to reflect the requirements for a plan of high quality. We show how to extract approximate solutions to the CSP using efficient clustering methods. We also include other considerations such as setup preferences, placement of datums on the CMM table, and segmentation of surface features through point sampling.

We decompose the high-level planning problem into knowledge acquisition and plan extraction. Knowledge acquisition involves the computation of domains for the variables in the CSP through the use of accessibility analysis. Plan extraction involves hierarchical constraint satisfaction through clustering techniques.

The system architecture is designed to postpone expensive geometric computations. Knowledge is acquired incrementally when needed, through lazy evaluation. Plans are extracted with incomplete knowledge, but are then verified with a validator. If the verification step fails, we incrementally acquire more knowledge and repeat the process.

If a valid HLIP is found, it is refined to a LLIP through path planning. The path planner linearizes the HLIP, while generating efficient paths through the measurement points. The path planner uses a roadmap method to connect the points through simple paths. It then makes use of existing TSP algorithms to find an efficient tour of the points in the roadmap.

6.2 Contributions

We supply a complete set of algorithms to compute global accessibility cones for straight and bent probes. The algorithms make use of computer graphics hardware, and therefore are very robust and efficient. Probes may be approximated realistically as a combination of grown half-lines.

We map the high-level inspection planning problem to a CSP. The result is a general theory for the existence of a HLIP and the requirements for a plan of high quality. Plans are extracted from the CSP using efficient clustering algorithms.

The planner architecture is novel and no backtracking is involved. By operating incrementally, it is often possible to produce a good plan without performing all

the geometric computations. The generate-and-test approach is effective, because the generation step is relatively smart. It takes into account the approachability constraints. When these constraints are (approximately) satisfied, the generated plan is highly likely to succeed.

We provide the first (probabilistically) complete path planner for CMMs. The planner is practical in the sense that CMM heuristics are easily integrated. Furthermore, existing TSP algorithms are used to find efficient paths.

The inspection planner has been implemented and includes an accessibility analysis module, a high-level planner, a plan validator (through collision detection), a simulator, and a path planner. We tested the planner on real-world mechanical parts and it is sufficiently fast for practical applications.

6.3 Limitations and Future Work

The accessibility analysis algorithms use a variety of abstractions and approximations that work very well in practice. One exception is the computation of D_2 (Section 2.5.2). This algorithm samples a set of points from a surface and computes the GAC at each point. This is an expensive process, unless a coarse sample is used. In practice, we postpone the computation of D_2 , and avoid computing it in many cases. However, sometimes it is necessary to compute D_2 , and an efficient algorithm would be useful. Unfortunately, we are not aware of such an algorithm.

The inspection planner is limited to CMMs with straight probes or bent probes. The theory of accessibility should be extended to star probes, that have several styli and tips. In addition, the planning problem could be extended to include additional sensors, such as laser range finders and cameras.

The high-level planner includes a mechanism for specifying setup preferences. Currently, we assume that prioritized lists of preferred setups are available a-priori. In the future, we should provide this information automatically through external agents, such as a stability analysis module and a fixturing module.

In our planner, we use a very simple clustering algorithm to extract HLIPs. The algorithm is greedy and does not consider preferences of setups and probes. Alternative algorithms, such as the one proposed in Section 3.6.1, may produce

better results. These algorithms need to be tested and compared empirically for efficiency, quality of results, and the flexibility provided to the user.

The system architecture is limited in two ways. First, the generate-and-test controller is very simplistic. The controller extracts a HLIP, finds the first invalid PIP (if one exists), constrains its domain, and repeats the process. A more sophisticated controller would balance the load between knowledge acquisition and plan extraction. Alternatively, one can use a parallel architecture to constrain the domains of variables in unison.

The second limitation is with human computer interaction. Our research focuses on a fully automated system. Therefore, we put little emphasis on the user interface. A real system should allow an operator to guide the planner in an intuitive and interactive process.

The goal of this work is to develop a fully automated dimensional inspection planner for CMMs. The planner has been implemented in a lab setting and tested with a simulator. Before pursuing a commercial planner it is necessary to execute the inspection plans on real CMMs and assess their performance against human generated plans. The performance criteria should include both planning time and plan quality.

Reference List

- [1] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning*. Morgan Kaufmann, 1990.
- [2] J. L. Ambite and C. A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, 1997.
- [3] ANSI. Dimensioning and tolerancing. American National Standard ANSI Y14.5M-1982, The American Society of Mechanical Engineers, United Engineering Center, 345 East 47 Street, New York, N.Y. 10017, February 1982.
- [4] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierarchies. In M. Jampel, E. Freuder, and M. Maher, editors, *Over-Constrained Systems*, number 1106 in LNCS, pages 23–62. Springer, August 1996.
- [5] J. A. Bosch, editor. *Coordinate measuring machines and systems*. M. Dekker, New York, 1995.
- [6] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Trans. Syst. Man Cybern.*, pages 190–197, 1983.
- [7] C. W. Brown. IPPEX: An automated planning system for dimensional inspection. *Manufacturing Systems*, 20(2):189–207, 1991. Proceedings of CIRP seminar in Computer Aided Process Planning, Enschede, Holland, June 11-12, 1990.
- [8] C. W. Brown and D. A. Gyorog. Generative inspection process planner for integrated production. In P. H. Cohen and S. B. Joshi, editors, *Advances in Integrated Product Design and Manufacturing, PED-Vol. 47*, pages 151–162. ASME, New York, 1990. Proceedings of ASME, Winter Annual Mtg., Dallas, TX, November 25-30, 1990.
- [9] J. Canny. *The Complexity of Robot Motion Planning*. ACM – MIT Press Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1987.
- [10] L.-L. Chen, S.-Y. Chou, and T. C. Woo. Separating and intersecting spherical polygons: Computing machinability on three-, four-, and five-axis numerically

- controlled machines. *ACM Transactions on Graphics*, 12(4):305–326, October 1993.
- [11] M. W. Cho and K. Kim. New inspection planning strategy for sculptured surfaces using coordinate measuring machine. *International Journal of Production Research*, 33(2):427–444, 1995.
 - [12] Computer Aided Manufacturing – International. *Dimensional Measuring Interface Specification Version 2.0*. CAM-I, Inc., April 1988.
 - [13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.
 - [14] M. J. Corrigan and R. Bell. Probe and component set-up planning for coordinate measuring machines. *International Journal of Computer Integrated Manufacturing*, 4(1):34–44, 1991.
 - [15] D. Dobkin and S. Teller. Computer graphics. In Jacob E. Goodman and Joseph O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 42, pages 779–796. CRC Press LLC, Boca Raton, FL, 1997.
 - [16] H. A. Elmaraghy and W. H. Elmaraghy. Computer-aided inspection planning (CAIP). In J. J. Shah, M. Mantyla, and D. S. Nau, editors, *Advances In Feature Based Manufacturing*, pages 363–396. Elsevier Science B. V., 1994.
 - [17] H. A. ElMaraghy and P. H. Gu. Expert system for inspection planning. *Annals of the CIRP*, 36, January 1987.
 - [18] K.-C. Fan and M. C. Leu. Intelligent planning of CAD-directed inspection for coordinate measuring machines. *Computer Integrated Manufacturing Systems*, 11(1-2):43–51, 1998.
 - [19] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.
 - [20] D. E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, October 1992.
 - [21] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 171–180. ACM SIGGRAPH, Addison Wesley, August 1996.
 - [22] J. B. Gou, Y. X. Chu, and Z. X. Li. On the symmetrical localization problem. *IEEE Transactions on Robotics and Automation*, 14(4):533–540, August 1998.

- [23] P. Gu and K. Chan. Generative inspection process and probe path planning for coordinate measuring machines. *Journal of Manufacturing Systems*, 15(4):240–255, 1996.
- [24] K. Gupta and A. P. Del Pobil, editors. *Practical Motion Planning in Robotics*. John Wiley and Sons Ltd, Baffins Lane, Chichester, West Sussex, PO19 1UD, England, 1998.
- [25] P. Gupta, R. Janardan, J. Majhi, and T. Woo. Efficient geometric algorithms for workpiece orientation in 4- and 5-axis NC machining. *Computer-Aided Design*, 28(8):577–587, 1996.
- [26] T. H. Hopp. CAD-directed inspection. *Annals of the CIRP*, 33, 1984.
- [27] T. H. Hopp and K. Lau. A hierarchical model-based control system for inspection. Special Technical Testing Publication 862, American Society for Testing and Materials, 1916 Race Street, Philadelphia, PA 19103, 1985.
- [28] J. Jackman and D.-K. Park. Probe orientation for coordinate measuring machine systems using design models. *Robotics and Computer-Integrated Manufacturing*, 14:229–236, 1998.
- [29] M. Jampel, E. Freuder, and M. Maher, editors. *Over-Constrained Systems*. Number 1106 in LNCS. Springer, August 1996.
- [30] R. Janardan and T. Woo. Manufacturing processes. In Jacob E. Goodman and Joseph O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 46, pages 851–862. CRC Press LLC, Boca Raton, FL, 1997.
- [31] L. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.
- [32] L. E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, Department of Computer Science, Stanford University, 1995.
- [33] B. Khoshnevis and Z. Yeh. An automatic measurement planning system for coordinate measuring machines. *Manufacturing Review*, 6(3):221–227, September 1993.
- [34] E. J. Klages and R. M. Wilson. APM: An automated inspection programming system. *International Journal of Computer Integrated Manufacturing*, 7(6):365–374, Nov-Dec 1994.
- [35] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, Spring 1992.

- [36] S. Kweon and D. J. Medeiros. Part orientations for CMM inspection using dimensioned visibility maps. *Computer-Aided Design*, 30(9):741–749, 1998.
- [37] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.
- [38] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. Wiley, New York, NY, 1985.
- [39] D. I. Legge. Integration of design and inspection systems: A literature review. *International Journal of Production Research*, 34(5):1221–1241, May 1996.
- [40] Z. X. Li, J. Gou, and Y. Chu. Geometric algorithms for workpiece localization. *IEEE Transactions on Robotics and Automation*, 14(6):864–878, December 1998.
- [41] C. P. Lim and C. H. Menq. CMM feature accessibility and path generation. *Robotics and Computer Integrated Manufacturing*, 32(3):597–618, 1994.
- [42] A. Limaïem and H. A. ElMaraghy. A general method for accessibility analysis. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 2346–2351, Albuquerque, New Mexico, April 1997.
- [43] A. Limaïem and H. A. ElMaraghy. A general method for analysing the accessibility of features using concentric spherical shells. *The International Journal of Advanced Manufacturing Technology*, 13:101–108, 1997.
- [44] A. Limaïem and H. A. ElMaraghy. Automatic path planning for coordinate measuring machines. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 887–892, Leuven, Belgium, May 1998.
- [45] Y.-J. Lin and P. Murugappan. A new algorithm for CAD-directed CMM dimensional inspection. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 893–898, Leuven, Belgium, May 1998.
- [46] Z.-C. Lin and C.-C. Chen. Measuring-sequence planning by the nearest neighbour method and the refinement method. *The International Journal of Advanced Manufacturing Technology*, 13:271–281, 1997.
- [47] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, February 1983.
- [48] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22:560–570, 1979.

- [49] E. Lu, J. Ni, and S. M. Wu. An algorithm for the generation of an optimum CMM inspection path. *Journal of Dynamic Systems Measurement and Control: Transactions of the ASME*, 116:396–404, September 1994.
- [50] A. D. Marshall and R. R. Martin. Automatic inspection of three-dimensional geometric features. *ASME Concurrent Engineering*, 59:53–67, 1992.
- [51] D. J. Medeiros, Thomas G., Ratkus A. B., and D. Cannon. Off-line programming of coordinate measuring machines using a hand-held stylus. *Journal of Manufacturing Systems*, 13(6):401–411, 1994.
- [52] C. H. Menq and H. T. Yau. An intelligent, computer-integrated and self-sustained environment for automated dimensional inspection of manufactured parts. In *NSF Design and Manufacturing Systems Conference, Austin, TX*, pages 865–872, January 9-11 1991.
- [53] C.-H. Menq, H.-T. Yau, and G.-Y. Lai. Automated precision measurement of surface profile in CAD-directed inspection. *IEEE Transactions on Robotics and Automation*, 8(2):268–278, April 1992.
- [54] F. L. Merat and G. M. Radack. Automatic inspection planning within a feature-based CAD system. *Robotics and Computer-Integrated Manufacturing*, 9(1):61–69, 1992.
- [55] F. L. Merat, K. Roumina, S. M. Ruegsegger, and R. B. Delvalle. Automated process planning for quality control inspection. U. S. Patent #5465221, November 1995.
- [56] G. Miller. Efficient algorithms for local and global accessibility shading. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 319–326. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
- [57] J. Mitchell. Shortest paths and networks. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 445–466. CRC Press LLC, Boca Raton, FL, 1997.
- [58] D. S. Nau, S. K. Gupta, and W. C. Regli. AI planning versus manufacturing-operation planning: A case study. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1010–1015, Seattle, WA, 1994.
- [59] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987.

- [60] J. O'Rourke. Visibility. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 25, pages 467–480. CRC Press LLC, Boca Raton, FL, 1997.
- [61] H. D. Park and O. R. Mitchell. CAD based planning and execution of inspection. In *Proceedings of the 1988 IEEE Computer Society Conference of Computer Vision and Pattern Recognition*, pages 858–863, 1988.
- [62] M. D. Reimann and J. Sarkis. Design for automating the inspection of manufacturing parts. *Computer Integrated Manufacturing Systems*, 7(4):269–278, November 1994.
- [63] G. Reinelt. The traveling salesman: Computational solutions for TSP applications. *Lecture Notes in Computer Science*, 840, 1994.
- [64] Renishaw Inc., 623 Cooper Court, Schaumburg, Illinois 60173. *Renishaw Product Catalog, Issue 3*, 1996.
- [65] A. A. G. Requicha. Representations for rigid solids: theory, methods and systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.
- [66] J. R. Rossignac, A. Megahed, and B.-O. Schneider. Interactive inspection of solids: Cross-sections and interferences. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 353–360, July 1992.
- [67] J. R. Rossignac and A. A. G. Requicha. Offsetting operations in solid modelling. *Computer Aided Geometric Design*, 3(2):129–148, August 1986.
- [68] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*,. Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-103805-2, 912 pp., 1995.
- [69] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 1.1). Technical report, Silicon Graphics, Inc., 1997.
- [70] Spatial Technology, Inc., 2425 55th Street, Suite 100, Boulder, Colorado 80301-5704. *ACIS 3D Toolkit: Technical Overview*, August 1997.
- [71] S. N. Spitz, A. J. Spyridi, and A. A. G. Requicha. Accessibility analysis for planning of dimensional inspection with coordinate measuring machines. IRIS Technical Report IRIS-98-360, Institute for Robotics and Intelligent Systems, University of Southern California, March 1998.
- [72] A. J. Spyridi. *Automatic Generation of High Level Inspection Plans for Coordinate Measuring Machines*. PhD thesis, University of Southern California, Department of Computer Science, August 1994.

- [73] A. J. Spyridi and A. A. G. Requicha. Accessibility analysis for the automatic inspection of mechanical parts by coordinate measuring machines. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1284–1289, Cincinnati, Ohio, May 1990.
- [74] A. J. Spyridi and A. A. G. Requicha. Accessibility analysis for polyhedral objects. In S. G. Tzafestas, editor, *Engineering Systems with Intelligence: Concepts, Tools and Applications*, pages 317–324. Dordrecht, Holland: Kluwer Academic Publishers, Inc., 1991.
- [75] A. J. Spyridi and A. A. G. Requicha. Automatic planning for dimensional inspection. *Manufacturing Review*, 6(4):314–319, December 1993.
- [76] A. J. Spyridi and A. A. G. Requicha. Automatic programming of coordinate measuring machines. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1107–1112, San Diego, California, May 1994.
- [77] X. Q. Tang and B. J. Davies. Integration of inspection planning system and CMM via DMIS. *International Journal of Advanced Manufacturing Technologies*, 10:422–426, 1995.
- [78] K. Tarabanis, R. Y. Tsai, and A. Kaul. Computing occlusion-free viewpoints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):279–292, March 1996.
- [79] E. Trucco, M. Umasuthan, A. Wallace, and V. Roberto. Model-based planning of optimal sensor placements for inspection. *IEEE Transactions on Robotics and Automation*, 13(2):182–193, April 1997.
- [80] A. Vafaeseefa and H. A. ElMaraghy. Accessibility analysis in 5-axis machining of sculptured surfaces. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 2464–2469, Leuven, Belgium, May 1998.
- [81] I. Walker and A. F. Wallis. Applications of 3-D solid modelling to coordinate measuring inspection. *International Journal of Machine Tools and Manufacture*, 32(1/2):195–201, 1992.
- [82] R. H. Wilson. *On Geometric Assembly Planning*. PhD thesis, Stanford University, Department of Computer Science, 1992.
- [83] R. H. Wilson. Geometric reasoning about assembly tools. *Artificial Intelligence*, 98(1-2):237–279, January 1998.
- [84] T. C. Woo. Visibility maps and spherical algorithms. *Computer-Aided Design*, 26(1):6–16, January 1994.

- [85] C. C. Yang and M. Marefat. Object-oriented concepts and mechanisms for feature-based computer-integrated inspection. *Advances In Engineering Software*, 20(2-3):157–179, 1994.
- [86] H.-T. Yau and C.-H. Menq. Path planning for automated dimensional inspection using coordinate measuring machines. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 1934–1939, Sacramento, California, April 1991.
- [87] H. T. Yau and C. H. Menq. An automated dimensional inspection environment for manufactured parts using coordinate measuring machines. *International Journal for Production Research*, 30(7):1517–1536, 1992.
- [88] H.-T. Yau and C.-H. Menq. Automated CMM path planning for dimensional inspection of dies and molds having complex surfaces. *International Journal of Machine Tools and Manufacture*, 35(6):861–876, 1995.
- [89] C. W. Ziemian and D. J. Medeiros. Automated feature accessibility algorithm for inspection on a coordinate measuring machine. *International Journal of Production Research*, 35(10):2839–2856, 1997.
- [90] C. W. Ziemian and D. J. Medeiros. Automating probe selection and part setup planning for inspection on a coordinate measuring machine. *International Journal of Computer Integrated Manufacturing*, 11(5):448–460, 1998.