Algorithms and Techniques for Manufacturing

by

Aaron Samuel Wallack

S.B. (Massachusetts Institute of Technology) 1990
M.S. (University of California at Berkeley) 1993

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor John F. Canny, Chair
Professor David Forsyth
Professor Max Mendel

1995

The dissertation of Aaron Samuel Wallack is approved:

Chair Date

Date

Date

University of California at Berkeley

1995

Algorithms and Techniques for Manufacturing

by

Aaron Samuel Wallack

Algorithms and Techniques for Manufacturing

by

Aaron Wallack

Abstract

This dissertation investigates the application of computer science techniques to tasks in manufacturing. Many of these techniques are founded upon three underlying approaches: machine vision techniques, generate and test techniques, and classical algebraic geometry techniques. The ten main algorithms presented in this thesis are: recognizing and localizing prismatic parts using a crossbeam sensor, recognizing and localizing prismatic parts using a scanning beam sensor, constructing complete indexing tables to recognize objects from crossbeam sensor data and scanning beam sensor data, localizing features of dissimilar types, enumerating all fixture configurations capable of immobilizing a given prismatic polyhedral object, enumerating all fixture configurations capable of immobilizing a given prismatic generalized polyhedral object, reducing the problem of enumerating modular fixture designs to the problem of constructing complete indexing tables, constructing complete indexing tables for generic systems, using tree grid structures to compact indexing tables, calibrating point probe and line probe sensors, and calibrating the end effector arm of high precision manipulators. We discuss the implementation of these algorithms and their performance on a number of examples. These techniques are also applicable to other problems in computer vision, and robotics.

The thesis consists of twelve chapters. Each chapter has been organized independently. Chapter 1 gives a brief overview of the thesis. In Chapter 2 we present the algorithm for recognizing and localizing prismatic parts using a crossbeam sensor, and discuss its implementation and performance. Chapter 3 describes the algorithm for recognizing and localizing parts using a scanning beam sensor and provide performance measurements. In Chapter 4, we present an exact algorithm for localizing features of dissimilar types by reducing it to a nonlinear least squares problem which is solved using a combination of algebraic and numerical techniques. In Chapter 5, we discuss a complete modular fixture design algorithm which efficiently enumerates all fixture configurations for immobilizing a

given prismatic polyhedral workpiece. In Chapter 6, we extend this algorithm to handle prismatic generalized polyhedral workpieces. In Chapter 7, we discuss the duality between modular fixturing and scanning which provides a generic algorithm for designing fixtures for arbitrary workpieces and arbitrary minimal fixture toolkits. In Chapter 8, we describe a technique for constructing complete indexing tables, thereby enabling indexing techniques to be used for systems which do not exhibit invariants. In Chapter 9, we describe the tree grid indexing mechanism which reduces the size of indexing tables, and also provides for coherent searches. In Chapters 10 and 11, we discuss calibration issues which are prerequisites for exploiting the high precisions achieved by the sensing strategies. Finally, in Chapter 12 we conclude the thesis and discuss problem areas for future research.

John F. Canny, Thesis Committee Chair

Contents

List of Figures

List of Tables

Acknowledgements

I have been incredibly fortunate to have had John Canny as an advisor throughout my graduate education. John has been a terrific teacher, and a great friend, and his varied knowledge and razor-sharp intuition have been great resources. Many of the ideas presented in this thesis were originally developed by John, or were born from fruitful discussion. His contribution to my work is immeasurable.

I would also like to thank Prof. Dinesh Manocha, for explaining resultants, elimination, and his code to me. Without his aid, most of the work presented in this paper would not have been possible. His perspective and insight have been extremely valuable. I'd especially like to thank him for the encouragement, and support that he's shown.

I would also like to thank multitudes of friends who spent inordinate amounts of time discussing ideas, and proofreading various drafts of this thesis. Prof. Ming Lin, Eric Paulos, Brian Mirtich, Dina Berkowitz, Ed Nicolson, Francesca Barrientos, Prof. Matthew Berkemeier, Prof. Ron Fearing, Prof. David Forsyth, Sanjay Sarma, Eden Tan, Prof. Paul Wright, Dr. Randy Brost, Trevor Blumenau, Prof. Ken Goldberg, Dr. Anil Rao, Dr. Michael Taylor, Yan-Bin Jia, and Yan Zhaung. I would also like to acknowledge Joe Gavazza and Ben Lake for designing and fabricating the devices I used: the crossbeam sensor, scanning beam sensor, and calibration peg. Finally, I would like to thank Milton Chen, Julie Chen, and Tyrone Nakahara for their assistance.

I could never have completed this thesis without the abundant love, support, and advice that my wife and family have given me throughout the years. My wife has been a tremendous source of inspiration and support, and without her, this work would not have been achievable.

Chapter 1

Introduction

Many tasks in manufacturing can be addressed by a combination of simple hardware and sophisticated algorithms. This combination addresses the specific needs of manufacturing: speed, robustness, cost effectiveness, and precision, and to a lesser extent, generality and flexibility. Although simple sensors and actuators are prevalent in industrial manufacturing, most of the research in automated manufacturing focus on more anthropomorphic approaches, such as camera-based machine vision, and knowledge engineering. Artificial intelligence approaches such as machine vision and knowledge engineering, which were developed for a broad class of problems do not address all of the issues of industrial manufacturing.

Specialized knowledge engineering systems and specialized machine vision systems comprise the majority of manufacturing automation. Still, even customization cannot completely overcome the limitations imposed by the underlying architecture. For example, automated fixture design expert systems usually are limited to following machining rules, although some systems have been extended to perform geometrical and mechanical computations. Similarly, machine vision systems rely on image data, which can be coarse, inaccurate, and requires significant computational overhead.

These limitations motivated us to address manufacturing tasks using novel approaches which relied on a combination of simple hardware and sophisticated algorithms. Simple hardware can provide efficiency, robustness, low cost, and the algorithms can provide generality and exploit the ever-increasing power and ever-decreasing cost of computation. This combination is widely applicable to other fields, including robotics, vision, and model construction. In this thesis, we present these novel approaches techniques, and demonstrate

their applicability towards industrial manufacturing tasks. As a consequence of using simpler devices, these approaches achieved high performance in terms of speed and accuracy, and the underlying algorithms were straightforward.

This thesis comprises two main sets of contributions: first, the techniques themselves, which are applicable to industrial applications, and second, example applications involving high-level computer science techniques. The main results of this research were generic indexing techniques used for object recognition, generic techniques for registering objects, and generic strategies for designing fixtures. The underlying theme of this research is the application of computer science techniques, such as computational geometry and computational algebra, to manufacturing problems. We reduced tasks to well understood problems, such as reducing the problem of constructing complete indexing tables to the problem of enumerating cells formed by an arrangement of curves. Five chapters (chapters 3,4,6,8,10) describe applications of resultants to solve multivariate systems using recent results by Manocha [Man92]. Four chapters (chapters 2,3,7,8) describe applications of the computational geometry concept of enumerating cells in an arrangement [Ede87].

The ten major new algorithms in this thesis solve the following problems:

- Recognize and localize modeled prismatic generalized polyhedral objects using sensors composed of crossed binary line probe sensors. This task can be accomplished in linear time without preprocessing, or in constant time via indexing. Indexing relies on construction complete indexing tables, and we describe a generic technique for systems with a single degree of freedom.

- Recognize and localize modeled generalized polyhedral objects using a scanning sensor composed of binary point probe sensors. This task can be accomplished in constant time via indexing. Indexing relies on construction complete indexing tables, and we describe a generic technique for systems with arbitrary configuration spaces.

- Localize a modeled object in the plane where the model features and the sensed features are of dissimilar types. The pose estimate which corresponds to the minimum squared error is computed exactly in constant time using a combination of algebraic and numerical methods.

- Design a modular fixture for holding and repeatably positioning prismatic polyhedral objects. This algorithm efficiently enumerates all valid fixture configurations for which

the fixture vice toolkit can immobilize a given object.

- Design a modular fixture for holding and repeatably positioning prismatic generalized polyhedral objects. This algorithm efficiently enumerates all valid fixture configurations for which the fixture vice toolkit can immobilize a given object.

- Design modular fixtures for arbitrary workpieces and arbitrary minimal modular fixture toolkits. This algorithm is based upon a duality between minimal fixtures and scanning. One consequence of this duality is that the task of enumerating fixture designs is reducible to the task of the constructing complete indexing tables for a related scanning sensing system.

- Construct complete indexing tables for the purpose of recognizing and interpreting modeled objects from sensed feature data in constant time. This method extends the complete indexing construction methodology to handle systems where multiple feature groups are sensed at each configuration.

- Compact indexing tables and exploit coherence by using tree grids for indexing, rather than a table lookup. This approach reduces the total size of the indexing tables so that more such tables can be combined in fast main memory.

- Calibrate point probe and line probe sensors from measurements made with respect to an unmodeled object. Effective calibration techniques are essential for achieving robust sensor techniques.

- Calibrate high precision manipulators by for measuring the offset of the end effector arms of four degree of freedom Robotworld and SCARA manipulators.

These algorithms are useful for problems in the following areas:

- **Computer Vision:** These techniques provide foundations for robust algorithms for solving the correspondence problem by constructing complete indexing tables, and global non-linear least squares problems found in localization, feature extraction, and structure from motion.

- **Robotics:** These techniques are applicable to the broader fields of integrating sensing and control, and grasp planning. Important problems in this area include localization for inspection, feature detection, and part presentation, and kinematic calibration.

1.1 Overview of the Thesis

This thesis presents algorithms and mechanical devices designed for tasks in manufacturing and other fields. The tasks involve object localization, object recognition, and fixture design. The rest of this thesis is organized in the following manner.

- Chapter 2 - **Crossbeam Sensing:** In this chapter, an efficient algorithm is presented to interpret data from crossed beam sensors to perform object recognition and localization. This recognition and localization technique relies upon simple sensors to extract necessary and sufficient data and recognize objects in 5 microseconds and localize objects repeatably to 25 microns. Our localization technique is based on reliable, highly accurate, robust, and inexpensive modulated LED light sources and detectors, which makes technique well suited for manufacturing because the components are inexpensive, precise, and easy to setup and calibrate. The technique involves moving generalized polyhedral objects relative to a crossbeam sensor, a set of coplanar oriented light beam sensors, and recording robot positions when the sensor's outputs change. Both object recognition and object localization share the same subproblems: the correspondence problem, the problem of interpreting the sensed features as model features, and the pose estimation problem, the problem of estimating the object's pose given an interpretation of the sensed features as model features. Since there are many satisfactory pose estimation techniques in the literature, we concentrate on the correspondence problem. In this chapter, we describe two methods for solving the correspondence problem for crossbeam data: a linear time completely online method, and a constant time online method which utilizes preprocessing

- Chapter 3 - **Scanning Beam Sensing:** In this chapter, an efficient algorithm is presented to interpret data from scanning beam sensors. We present recognition and localization techniques and experimental results which use a scanner composed of binary light beam sensors to quickly recognize objects (as fast as 5 microseconds) and to accurately localize objects (0.025 millimeters). The binary scanning sensors detect only whether the part obstructs a light beam. The sensors high performance in terms of speed and accuracy is a consequence of the sensor's simple specification. Our scanning sensor is reliable, inexpensive, compact, precise, and insensitive to ambient light, which are all prerequisites for industrial manufacturing. Objects are scanned by

passing them through a curtain of light beam sensors, and recording the position where the sensor outputs changed. Interpreting the resulting data involves matching the sensed data to model features, which is termed the correspondence problem; we solve the correspondence problem via a constant-time indexing technique which involves looking up the interpretation in a precomputed indexing table.

Indexing is a general approach to solving the correspondence problem in which all predicted interpretations are stored in a table indexed by the sensor data. Complete indexing tables are crucial for sparse sensing strategies such as scanning beam sensing because each experiment produces only a single indexing vector. In this chapter, we describe a method for constructing complete indexing tables by reducing it to the task of enumerating all cells formed by an arrangement of correspondence boundaries discretization boundaries.

- Chapter 4 - **Localization from Dissimilar Features:** This chapter considers the problems of localizing modeled features from sensed features of a different type, such as transforming a set of points onto a set of linear features. Object localization is reformulated as a least squares problem: the optimal pose estimate minimizes the squared error (discrepancy) between the sensed and predicted data. The resulting problem is non-linear and previous attempts to estimate the optimal pose using local methods such as gradient descent suffer from local minima and, at times, return incorrect results. This chapter describes an exact, accurate and efficient algorithm based on resultants, linear algebra, and numerical analysis, for solving the nonlinear least squares problem associated with localizing two-dimensional objects given two-dimensional data. This work is aimed at tasks where the sensor features and the model features are of different types and where either the sensor features or model features are points. It is applicable to localizing modeled objects from image data, and estimates the pose using all of the pixels in the detected edges. Previous approaches have sampled points on the model feature to match or limited their scope to matching interesting features such as vertices; this technique computes the error between the sensed data points and the model features, enabling all boundary points to be utilized. The algorithm's running time depends mainly on the type of non-point features, and it also depends to a small extent on the number of features. On a SPARC 10, the algorithm takes a few microseconds for rectilinear features, a few milliseconds for

linear features, and a few seconds for circular features.

- Chapter 5 - **Automatically Designing Modular Fixtures:** Fixturing encompasses the design and assembly of fixtures to locate and hold a workpiece during a manufacturing operation such as drilling or assembly. Automated fixture design is very difficult due to the open-endedness of the problem. Fixture quality depends not only on the *configuration*, and the expected process operations, by also upon the mechanical characteristics of the fixturing elements in the fixture kit; a fixture kit of simple modular mechanical components can be more precise, less expensive, and more robust than a fixture kit of more complicated components. An automated fixture design algorithm was implemented for a modular fixturing kit consisting of a fixture vice and modular fixturing elements (pegs or flatted pegs). Generically, an appropriately sized fixture vice system allows many configurations for immobilizing a two-and-a-half dimensional workpiece. In this chapter, we present an algorithm which, given a workpiece, enumerates all force closure fixtures, where the term force closure describes the capability of resisting arbitrary forces and torques. The algorithm runs in $O(A)$ time, where A is the number of configurations achieving simultaneous contact between a model and four fixture elements; since simultaneous contact is a necessary, yet not a sufficient precondition for force closure, A provides an upper bound on the number of force closure fixture designs. This algorithm can be used as part of a higher level planner, which, given a set of workpieces, determines an operation sequence and fixtures to reuse fixtures and minimize the total fixture changeover time.

- Chapter 6 - **Fixture Design for Generalized Polyhedra:**

  In this chapter, we extend the fixture vise configuration algorithm to handle generalized polyhedral prismatic workpieces; the term generalized polyhedral prismatic workpiece refers to a workpiece which, in its fixture pose, has a generalized polygonal projection, where a generalized polygon refers to a planar object having a boundary composed of linear and circular arcs.

- Chapter 7 - **The Duality Between Modular Fixturing and Scanning Sensing:**

  Fixturing, the act of immobilizing a workpiece for manufacturing or assembly operations, is a fundamental task in manufacturing. Fixtures can be fabricated from scratch or assembled from a toolkit of modular components; the latter approach is

termed modular fixturing. Recently. algorithms have been presented which automatically design fixtures for arbitrary objects. In this paper, we generalize these results into a generic strategy for enumerating modular fixture configurations for arbitrary workpieces and fixture toolkits, where the fixture toolkit consists of fixture elements which are inserted into a lattice of holes, and where the total number of contacts is equal to the total number of degrees of freedom. For these types of toolkits, there are only a finite number of configurations achieving simultaneous contact between the workpiece and the fixture elements, and, therefore, a generate and test approach is adequate. Our generic fixture design algorithm generates configurations which achieve simultaneous contact between the object and the fixture, and then prunes away configurations which do not exhibit force closure. This general strategy is based upon a duality we observed between modular fixturing and scanning sensing; it turns out that every modular fixture configuration is analogous to a scanning sensing situation. We arrived at our generic fixture design algorithm by extending a generic algorithm for the related problem of identifying objects from scanning sensors.

- Chapter 8 - **Construction of Complete Indexing Tables:**

  In this chapter, we present an algorithm for constructing complete indexing tables which broadens the applicability of indexing to non-dense sensing systems which do not exhibit invariance. Using this technique, we implemented a recognition system which identifies polyhedral objects from unscaled orthographic projective image data. The sensed feature groups consist of pairs of rays (pairs of edges which share a coincident vertex); five indexing coordinates are extracted from each pair of rays: four orientation measurements and one length measurement. These coordinates are quantized to achieve integral indices for table lookup. In this paper, we describe our technique for constructing complete indexing tables and present sizes of complete indexing tables for various objects and table discrtizations.

- Chapter 9 - **Tree Grid Indexing:** In this chapter, we investigate the performance of using tree grid structures to perform indexing, which has two advantages over hash tables: (i) The tree grid preserves spatial ordering, so that nearby indexing entries can be retrieved efficiently (ii) The tree grid compacts the storage size of the table by a factor of as much as two orders of magnitude. k coordinates index an ordering of the interpretations, and 1 coordinate determines the consistent interpretations for objects

with k degrees of freedom. We also show that for almost all model sets, $2k+1$ indexing coordinates are sufficient to discriminate between two generic models, implying that $2k+1$ indexing coordinates specify a unique interpretation. We have implemented an indexing algorithm for recognizing 3D objects from pairs of image rays using the tree grid technique, and the results are reported.

- Chapter 10 - **Calibrating Point and Line Probe Sensors:** In this chapter, we present a technique for calibrating point and line probe sensors. Simple binary sensor systems are capable of handling many industrial tasks currently handled by machine vision systems, and simple sensors have higher performance and are usually easier to set up and calibrate than complex sensors due to their simpler specifications. In this chapter, we present calibration techniques for four different types of sensors: mobile and stationary point scanning and line probing sensors. These calibration methods involve enumerating robot configurations for which a binary sensor detects an object. The sensor's position is determined from the robot configurations using a combination of algebraic and numerical methods. We observed two dualities intrinsic in these computations allowing us to use a single numerical solver routine to compute the sensor position for all four different sensor types. Using these techniques, we calibrated various sensors and then measured the accuracy of these sensor position estimates using cross validation techniques. We found sensor calibration accurate to 0.025 millimeters.

- Chapter 11 - **Calibrating End Effector Arms** In this chapter, we describe a technique which calibrates end effector arms of Robotworld modules, a prerequisite for the coordination of disparate devices. Previous research in calibration focused predominantly on difficult-to-model mechanically-complex anthropomorphic arm-based robots. Instead, we focus on a Robotworld module, a simple four degree of freedom (x, y, z, θ) manipulator designed for vertical assembly operations. We chose Robotworld modules because they combine high performance and simple calibration. As a consequence of Sawyer motor actuation, all of the modules are implicitly coordinated in x and y. Thereby, calibration only involves modeling the end effector arm's axis of rotation and axis of translation. We estimate these axes by localizing calibration peg via multiple crossbeam sensors at various heights. This technique coordinates Robotworld end effectors to 0.025 millimeters, as compared to 1 millimeter without

kinematic modeling.

- Chapter 12 - **Conclusion:** We conclude by restating the major results in this thesis and proposing areas for future research.

Chapter 2

Generalized Polyhedral Object Recognition and Localization Using Crossbeam Sensing

Abstract

Object localization is a fundamental problem in industrial automation. We present a recognition and localization technique which relies upon simple sensors to extract sufficient data and recognize objects in 5 microseconds and localize objects repeatably to 25 microns. Our localization technique is based on reliable, highly accurate, robust, inexpensive, and easy to calibrate modulated LED light sources and detectors, making this technique suitable for manufacturing. The technique involves moving generalized polyhedral objects relative to a crossbeam sensor, a set of coplanar oriented light beam sensors, and recording robot positions when the sensor's outputs change. Both object recognition and object localization share the same subproblems: the correspondence problem, the problem of interpreting the sensed features as model features, and the registration problem, the problem of estimating the object's pose given an interpretation of the sensed features as model features. We concentrate on the correspondence problem because there are many reliable pose estimation techniques found in the literature. In this chapter, we describe two methods for solving the correspondence problem for crossbeam data: a linear time completely online method, and a constant time online method which utilizes preprocessing.

2.1 Introduction

Current robotic hardware systems combine high speed and high accuracy, but exploiting these capabilities requires precise information about the environment. Precise assembly and machining tolerances also require high precision information. Nevins and Whitney found that simple peg-in-hole insertion tasks were the most frequent assembly operation [NW78], and such operations may have tolerances on the order of a few thousandths of an inch. Machine vision systems are commonly used to acquire accurate positional information, but these systems are slower and perhaps less precise than the robotic hardware.

The term *model based object recognition* refers to the task of determining which object O from a set of candidate objects $O_1, O_2, \ldots$ best explains the sensed data. The term *model based object localization* refers to the task of determining the pose p for a given modeled object O which best explains the sensed data. In principle, any model based recognition or localization task can be handled by a machine vision system, but strategies based on simpler sensors can attain higher performance in terms of speed, accuracy, reliability, and ease of use and setup. Commercially available vision systems are capable of identifying and localizing flat objects to arbitrary precision using lenses, but such systems require precise calibration and nontrivial computational hardware.

In this chapter, we present a recognition and localization technique which involves passing an object through a crossbeam sensor and recording robot positions when the sensor's outputs change; a crossbeam sensor is composed of a set of coplanar oriented light beam sensors (Figure 2.1). By extracting sufficient information, recognition and localization can be performed as fast (≈ 0.1 seconds) and as accurately (≈ 25 microns) as high speed manufacturing requires. This technique was specifically developed for industrial applications; its binary light beam sensors provide robustness, speed, reliability, and precision, which are all prerequisites for manufacturing applications.

2.1.1 RISC Robotics

This work is part of a larger research agenda termed RISC (Reduced Intricacy Sensing and Control) Robotics which attempts to solve problems using a combination of simple hardware and sophisticated software. Canny and Goldberg use the acronym RISC to serve as a focal point for research in simple sensing and control techniques [CG94]. This line of research was motivated by performance gains achieved in RISC microprocessor design

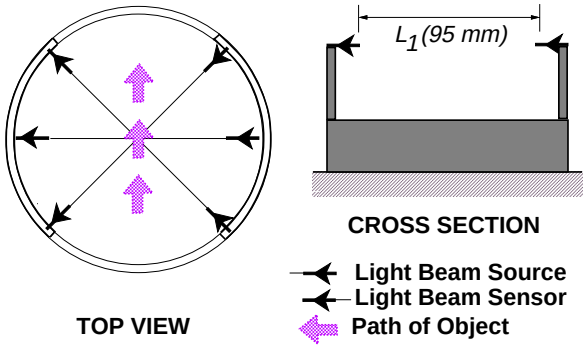


Figure 2.1: Crossbeam sensor apparatus.

by simplifying the instruction set. Instead of using a single multipurpose sensor, the RISC paradigm favors using many different *specialized* sensors, because specialized sensors and algorithms can be fast and robust. Furthermore, simple sensors are easy to setup and calibrate; setting up a light source and calibrating a camera takes much longer than installing a simple crossbeam sensor. Both model-based object recognition and object localization share the same difficult problem: the correspondence problem, the task of enumerating valid interpretations of sensed features as model features. Generally, an efficient solution to the correspondence problem goes a long way towards the larger tasks of object recognition and object localization. Object recognition involves verifying each of the candidate interpretations enumerated by the correspondence algorithm, by finding a pose for which the predicted data agrees with the sensed data. Object localization involves using the interpretations to determine the pose for which the predicted data optimally matches the sensed data. In this chapter, we focus on the correspondence problem because there are many pose estimation approaches found in the literature.

In this chapter, we describe algorithms for solving the correspondence problem for crossbeam sensor data. The crossbeam sensor only perceives a two-dimensional cross-section of the object, and our correspondence algorithms are designed for generalized polygonal cross-sections, cross-sections whose boundaries are composed of linear edge and circular arc features. Thereby, this system handles generalized polyhedral objects which are stably resting on a face of their convex hull such that the crossbeam sensor perceives a generalized polygonal cross section. Generalized polyhedral objects are defined to be volumes whose boundaries are composed of planar, cylindrical, and spherical faces.

2.1.2 Previous Work

This work is related to various topics in robotics and machine vision: light beam sensing, line probing, the correspondence problem, and pose estimation. Light beams and laser beams have been commonly used in order to achieve high performance. The crossbeam sensor is related to line probing because each beam sensor essentially functions as an oriented line probe. Furthermore, the more general problems of determining correspondences and estimating poses have been discussed at length in the machine vision literature and robotics literature.

Many research projects have utilized simple light beam sensors for high precision applications. Wallack [Wal95c] described an exact constant-time method for calibrating simple binary beam sensors, and experimentally determined that 40 measurements were sufficient to localize the beams to an accuracy of 10 microns. Wallack [Wal95a] used two crossbeam sensors to calibrate Robotworld manipulators [ATT67], and reported absolute positioning accuracy of 25 microns. Everett and Ives [EI93] use a crossed beam sensor apparatus for robot calibration. Prenninger *et al.* [PVG93] and Newman and Osborn [NO93] described a high-precision laser-beam based calibration technique. Lee and Hahn [LH90; LH91] determined the optimal sensing strategy for part localization using proximity sensors. Hong and Kleeman [HK92] used a triangular array of ultrasonic sensors to identify planes, edges, and corners. Wallack and Canny described a recognition and localization technique using a scanning beam sensor [WC93]. Paulos and Canny demonstrated 100% success in 100 trials at peg-in-hole insertion with a 25 micron clearance by using complementary crossbeam sensors and scanning beam sensors [PC93; PC94]. The main distinction between this work and most of the previous beam systems is that most of the previous work only performed localization whereas our system performs both recognition and localization.

One of the most difficulties of identifying objects from sparse beam data is determining the model features corresponding to the sensed data. The correspondence problem has been addressed both in the context of the robotics literature and the machine vision literature; in the machine vision literature, the model-based correspondence problem involves interpreting image features as model features, and we will shortly examine these approaches in more detail. In the robotics literature, this problem has been addressed explicitly, in terms of recognition and localization, and implicitly, in sensorless manipulation. Many robotic systems have recognized and registered (localized) modeled objects using a

variety of sensing modalities including: finger gap sensing, line probe sensing, etc. Most systems are feature based, and solve the correspondence problem in one form or another. In addition, the correspondence problem also appears implicitly in sensorless manipulation strategies, where the goal is to construct a plan which achieves the desired goal irrespective of the initial configuration without sensing.

Our work in crossbeam sensing stems from previous recognition work in finger gap sensing in which objects are grasped by a parallel-jaw gripper, and the distance between the jaws is used to identify the object. Goldberg and Mason [GM90] introduced the *diameter function* [PS85] to the robotics community to model the distance between parallel jaw grippers, to predict the finger gap distances. Goldberg *et al.* [KG93; RG92; RG93; GM90; GM91] have also developed systems to recognize objects using multiple finger gap measurements furnished by multiple parallel jaw grasping operations. In this case, the correspondence problem involves determining the which model features are contacting the parallel jaw grippers given the finger gap measurements. Our crossbeam sensor was motivated by this approach, with the crossed beam sensors characterizing three different finger gap measurements.

Crossbeam sensing is also related to line probing, which involves sweeping a line through an object and determining when the line first and last contacts the object. These two topics are related by virtue of the fact that each binary beam sensor in the crossbeam sensor functions as a line probe. Identifying objects from line probe data poses an interesting problem because the probe data does not specify a specific contact point, but rather a line of possible contact points. Cole and Yap [CY87] presented linear time algorithms for constructing polygonal object models using line probes, and for recognizing and localizing polygonal objects using line probes. The main distinction between generic line probing and crossbeam sensing, is that in crossbeam sensing, the probe orientations are fixed, whereas, for generic line probing, the algorithm chooses the orientation of the line probes.

The correspondence problem is also implicitly addressed in sensorless manipulation strategies. Erdmann and Mason [EM86] developed a technique for uniquely orienting an allen wrench via tray-tilting operations; their approach characterized each configuration by a node in a transition graph, where the edges in the graph specified transitions corresponding to tray-tilting operations. They developed an algorithm which determined operation sequences which were guaranteed to attain the desired final configuration regardless of the initial state. Goldberg *et al.* [Gol92; RG92] developed a different, though related,

technique which uniquely oriented prismatic parts (up to symmetry) via parallel-jaw grasps and push-grasp operations. Both approaches were implicitly concerned with the correspondence problem by virtue of the fact that they focused on a finite configuration space which characterizes which model features contact operative features.

The correspondence problem has been addressed at length in the machine vision literature, which is concerned with extracting information about a three dimensional scene from a two, or three, dimensional image data. The main difference between solving the correspondence problem for crossbeam sensor data, and solving the correspondence problem for camera data, is that the crossbeam sensor provides only a handful of highly precise data values, whereas cameras provide a large amount of medium precision data. This distinction suggests that correspondence algorithms specifically designed for machine vision applications may not be optimal for crossbeam sensing.

In the machine vision literature, there are two main approaches for solving the correspondence problem: the generate and test paradigm, and the indexing paradigm. The basic tenet of both approaches is that the object's pose is determinable after interpreting only a fraction of the sensed features, thus constraining the interpretations of the remaining features; *i.e.* interpretations of the remaining sensed features are either consistent or inconsistent with the interpretation of the first k sensed features. Any entirely self-consistent interpretation is assumed to be valid; often, only a single interpretation is completely self-consistent.

2.1.3 Alignment Overview

The generate and test paradigm involves enumerating hypothetical interpretations of the sensed features, and validating the self-consistency of these hypotheses. This is accomplished by comparing every sensed feature group with every possible model feature group. Indexing techniques, on the other hand, use overconstraining data from sensed feature groups and directly look up the consistent model feature group interpretations.

Without an effective heuristic, the combinatorial explosion of matching sensed feature groups to model feature groups takes time exponential in the group size since there are an exponential number of possible interpretations. In generate and test techniques, there are three main heuristics for pruning the inconsistent hypotheses: the alignment method, the pose clustering method, and the interpretation tree method.

In alignment methods, the object's pose is determined by computing the transformation(s) which map a group of k model features into alignment with a group of k sensed features [Rob63; FB81]. The hypothetical poses are then validated by predicting the remaining model features and comparing them with the remaining sensed features. Huttenlocher and Ullman [HU90] gave an efficient and accurate algorithm for computing the poses of a three-dimensional object by aligning sensed feature groups (consisting of three sensed points) with model feature groups (consisting of three points). Their alignment algorithm ran in $O(n^3m^3v)$ time ($k = 3$) for point groups, or $O(n^2m^2v)$ time ($k = 2$) for image ray groups (each ray characterizes a pair of coincident edges), where n is the number of observed image features, m is the number of model features, and v is the amount of time spent validating each hypothesis. These running times can be reduced to $O(nm^k \log(n))$ [Ols94b] via randomization techniques by tolerating a small fraction (δ) of false negatives.

Pose clustering methods are similar to alignment methods in that they also involve computing the object's pose from a group of k model features and a group of k sensed features [Hou62; Bal81]. Thompson and Mundy used pose clustering methods to identify polyhedral objects from image data by computing the transformations which mapped model vertex pairs into image vertex pairs, and histogramming these transformations to determine the object's actual pose [TM87]. The main distinction between pose clustering and alignment is that in pose clustering, all of the hypotheses are verified after they have all been postulated; in alignment, each hypothesis is validated immediately after it is postulated. Pose clustering algorithms maintain histograms characterizing the predicted poses, and then the hypotheses are verified in order of decreasing popularity. Assuming that the correct poses correspond to peaks in configuration space, pose clustering techniques have running times of $O(m^k n^k + v)$. Again, these running times can also be reduced to $O(nm^k)$ [Ols94b] via randomization techniques by tolerating a small fraction (δ) of false negatives.

Interpretation tree methods differ from alignment and pose clustering in that they do not explicitly compute the object's pose. Rather, the problem of finding a consistent interpretation is reformulated as a search tree problem, with each node characterizing an interpretation of a sensed feature as a model feature or *nil*. Unary and binary incompatibilities are used to intrinsically prune the search [FH86; Gri86]. In terms of algorithmic complexity, Grimson showed ($O(2^n)$) worst case recognition performance where the image contained n possibly spurious image features [Gri88].

One of the major drawbacks of generate and test techniques is that we have to check each combination of sensed feature groups and model feature groups. In other words, if the scene contains 100 sensed feature groups, and the model database contains 100 model feature groups, we would need to perform 10,000 alignment operations. We would greatly prefer a technique which simultaneously compares the sensed feature group to all of the model feature groups, and return all of the consistent model feature groups, because this would entail only 100 operations. Indexing is one technique which achieves simultaneously compares the sensed feature group with all of the model feature groups.

2.1.4 Indexing Overview

Indexing techniques are fundamentally different in that indexing coordinates are distilled from sensed feature groups and are quantized, and are used to index a table entry containing all the valid interpretations of the sensed feature group. Indexing relies on the assumption that (noisy) indexing coordinates from actual measurements are discretized into the same or nearby table entry as the predicted indexing coordinates. Furthermore, indexing relies upon the fact that only a small fraction of all indexing coordinates is consistent with each model feature group; otherwise, a large fraction of model feature groups may match a sensed feature group, which will reduce the performance of the technique to a brute force generate and test approach. Indexing is a robust technique applicable to various problems, and indexing tables have been used to recognize two-and-a-half dimensional objects from two-dimensional images [AF86], three dimensional objects from two-dimensional images [Ols94a], and two or three dimensional objects from two or three dimensional data [LSW88].

Indexing techniques are fundamentally different from generate and test techniques in that they use overconstraining feature groups of size $> k$. Therefore, only a small fraction of indexing coordinates should be consistent with each model feature group. Consequently, indexing tables for many model feature groups can be merged into a composite indexing table with the advantage that a single lookup operation efficiently compares the sensed feature group with all of the model feature groups. In this way, indexing techniques trade off complex offline computations for simple online operation.

Indexing techniques have mainly been used in conjunction with geometric invariant properties, properties which remain true regardless of viewpoint and object pose, because,

in these cases, each model feature group can be characterized by a single indexing table entry [KSSS86; SS87; FMZ+91]. Lamdan and Wolfson [LW88] distilled values from a model feature group consisting of three points which were invariant with respect to rigid planar transformations and scaling. Unfortunately, it has been shown that there are no invariants for an arbitrary number of three-dimensional points for orthographic [CJ91] or perspective projection [BWR93].

Pose estimation is the final subproblem involved in recognizing and localizing objects from crossbeam sensor data. Most of the approaches found in the literature were developed for image data, where there is an overabundance of data points. Kalvin *et al.* developed an efficient and robust method for computing the rigid transform which optimally maps one point set onto another [KSSS86; SS87; FH86]. Even when the sensed features are not in a one to one correspondence with model features, the non point features can be sampled to achieve point to point correspondences. Crossbeam sensing, on the other hand, only provides a handful of data points, and we therefore cannot afford the luxury of sampling data. We therefore use the localization technique described by Wallack and Manocha [WM94b] which finds the optimal pose estimate for point features matched to linear features.

2.1.5 Overview

In this chapter, we present two methods for solving the correspondence problem from crossbeam data, a completely online method and an online method utilizing preprocessing. Objects are sensed by moving an object relative to a crossbeam sensor, a set of coplanar, oriented light beam sensors, as shown in Figure 2.1. The robot's position is recorded at break points, when a light beam sensor first breaks or finally reconnects (Figure 2.2). The crossbeam sensor only detects the object's cross section in the beam plane. Although we cannot guarantee that such a simple sensor can uniquely identify an object, we can estimate the probability that passing an object through the crossbeam sensor will result in a unique interpretation.

The accuracy of this technique depends upon the accuracy of resolving the manipulator's position and the straightline motion (uniform orientation) of the manipulator. The path through the crossbeam sensor is irrelevant because we are only concerned with the break points. Maintaing constant orientation of the object is critical because it represents a

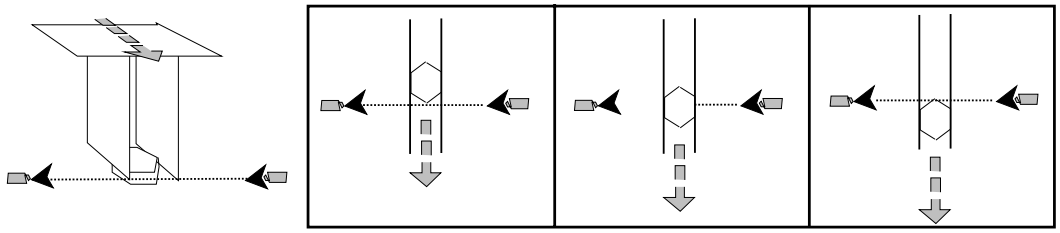
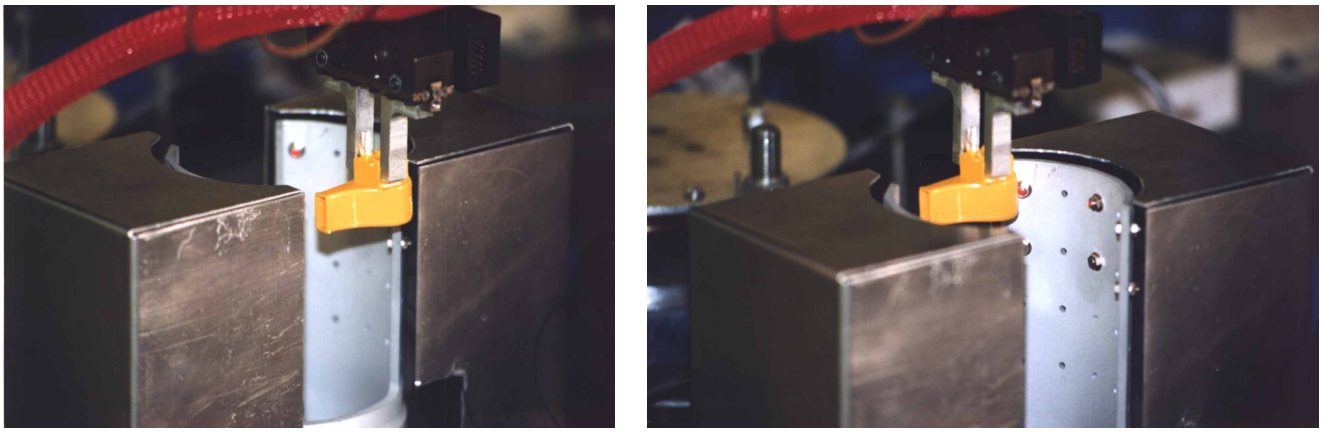


Figure 2.2: *Break* points occur when the light beam is broken or reconnects.



(a) (b)

Figure 2.3: Objects are recognized and localized after passing through the crossbeam sensor.

vital assumption for the indexing strategy, although variable orientations can be handled by the online correspondence algorithm. Robotworld, which relies on linear stepper (Sawyer) motors for actuation provides high precision and uniform orientation, making it ideal for crossbeam sensing.

Another advantage of this technique can localize objects during transport by placing multiple sensors throughout the workspace, along every path from part feeders to assembly; performing localization-during-transport with a machine vision system could require significant complexity (strobe lights, etc.)

Since the crossbeam sensor only produces a small set of high precision values, this sensor cannot perform useful tasks without imposing additional constraints. The localization technique requires that the object be initially resting on a flat surface of its convex hull; therefore, the crossbeam sensor will always perceive generalized polygonal cross sections of generalized polyhedra, and since there are a finite number of bottom faces, we only need to consider a finite number of generalized polygonal cross sections. The algorithm fails when:

1. The object is not stably resting on a flat surface.

2. Objects are not singulated (presented one at a time); *i.e.*, the breakpoints correspond to more than one object.

3. The object to be identified is flat or translucent; *i.e.*, undetectable by the light beam sensors.

4. Unique identification is impossible because two models have cross-sections with equivalent convex hulls.

The main contributions of our work are a linear-time online correspondence algorithm, and a constant-time correspondence algorithm which utilizes preprocessing. In this chapter we describe a method for computing complete indexing tables. We extract translationally-invariant data so that we only have to deal with a relatively simple single degree of freedom (θ) problem.

In this chapter, we present a completely online linear time correspondence algorithm similar to pose clustering. Since the system only provides a single degree of freedom, each measured diameter and interpretation specifies a small set of consistent orientations. Assuming model and sensor uncertainty each model feature is only consistent with two

ranges of orientations. The algorithm involves computing the configurations (θ orientation ranges) consistent with each of the three diameters, and efficiently computing the orientation(s) which are consistent with all sensed diameters by scanning down θ.

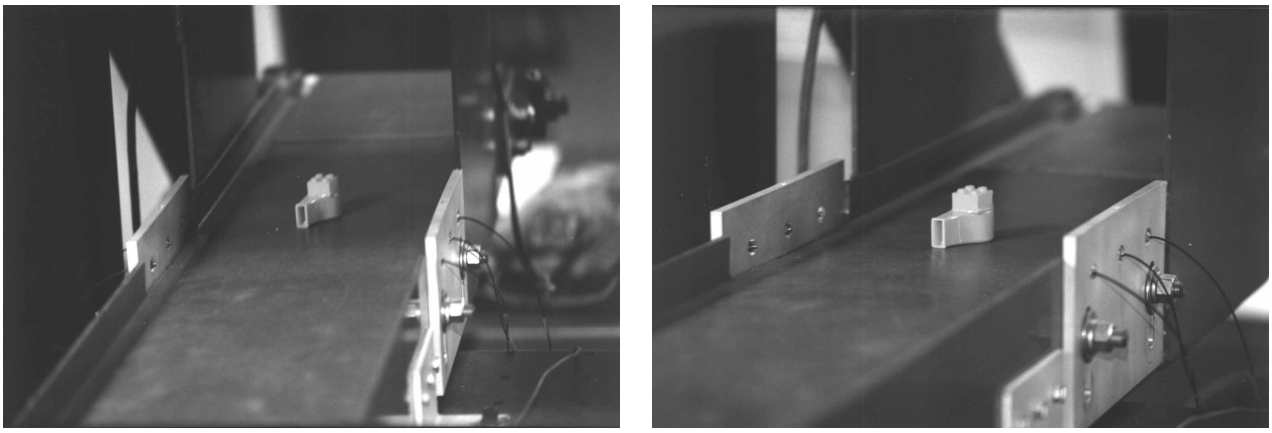
In this chapter, we also present a constant time indexing approach to the correspondence problem which relies on offline preprocessing. Indexing for crossbeam sensing is inherently different from indexing for machine vision applications because machine vision data contains a plethora of sensed features whereas crossbeam sensing only provides a handful of measurements. For machine vision techniques, where image data contains many sensed feature groups, correct operation requires only that the table contain one of the correct table entries corresponding to a sensed feature group. For crossbeam sensing, on the other hand, since each experiment only provides a single indexing vector, complete indexing tables are a prerequisite for correct operation. For systems which exhibit invariants, constructing complete indexing tables is a trivial task, but for non-degenerate cases, previous researchers have oversampled, or randomly sampled configuration space. Since the system has a single degree of freedom, the indexing coordinates trace out a one-dimensional curve in a higher dimensional indexing space. We construct complete indexing tables by walking along the curve of indexing coordinates and enumerating all of the indexing table entries.

In section 2, we explain how objects are perceived by crossbeam sensors, and we outline how the object's pose is estimated given the interpretation provided by the correspondence algorithm. In section 3, we present theoretical background and geometrical framework for both correspondence algorithms. In section 4, we present the online correspondence algorithm, and in section 5, we present a constant time indexing approach for solving the correspondence problem in constant time. Crossbeam sensor calibration the crossbeam sensor is discussed in section 6. We step through an example in section 7, and present experimental results in section 8. We conclude by highlighting the results and advantages of this technique.

2.2 Crossbeam Perception

In this section, we explain how the crossbeam sensor perceives objects. The crossbeam sensor's coplanar beams are only capable of sensing the object's cross section in the beam plane, and since they are beam sensors, they function as oriented line probes. Therefore, the crossbeam sensor perceives the enclosing parallelepiped of the object's two-

dimensional cross-section in the beam plane. Identifying object from such data is relatively simplified because the parallelepiped's geometry only depends upon the object's orientation and is independent of the object's position.



(a) (b)

Figure 2.4: Objects are identified and localized while moving along a conveyor belt.

Crossbeam sensing involves moving the part through the sensor, and recording breakpoints which describe the actuator positions when the beam sensor outputs changed. Given the breakpoint data, the localization technique computes the object's position with respect to a reference frame which moves with the object. This specification is general enough general to handle various situations: passing objects through a stationary crossbeam sensor, and passing a mobile crossbeam sensor around an object, transporting the objects using a robotic system, transporting the objects via a conveyor belt (Figure 2.4).

The crossbeam data is interpreted as line probe constraints. Figure 2.5 shows an object (held by a robot end-effector) moving in the y direction through a horizontal light beam sensor. The resulting break point data is of the form: *Beam i occluded when robot end-effector was at* $(Y = y_{i,0})$*; Beam i reconnected when robot end-effector at* $(Y = y_{i,1})$. Given such data, the task is to efficiently identify and localize the sensed object; *i.e.*, identify and determine the object's position in the end effector reference frame.

Performing this task requires introducing two abstractions: virtual break points and relative break points. The term virtual break point (Y_{Ref} in Figure 2.5(a)) refers to the manipulator's position when the origin of the reference frame would occlude the beam, and a relative break point is the normalized breakpoint obtained by subtracting the measured

break point from the virtual break point. Relative break points describe the line probe constraints with respect to the reference frame.

In Figure 2.5, the two break points $Y_{Ref}+1$,$Y_{Ref}+4$ correspond to relative break points of $\Leftrightarrow$1, $\Leftrightarrow$4. Relative break points provide line probe constraints with respect to the reference frame: for an object to have occluded the beams at position $Y_{Ref}+1$, the object must contact and lie completely below the line $y = \Leftrightarrow$1 in the end effector reference frame, and for an object to have occluded the beams at position $Y_{Ref}+4$, the object must contact and lie completely above the line $y = \Leftrightarrow$4 in the end effector reference (Figure 2.5(b)).

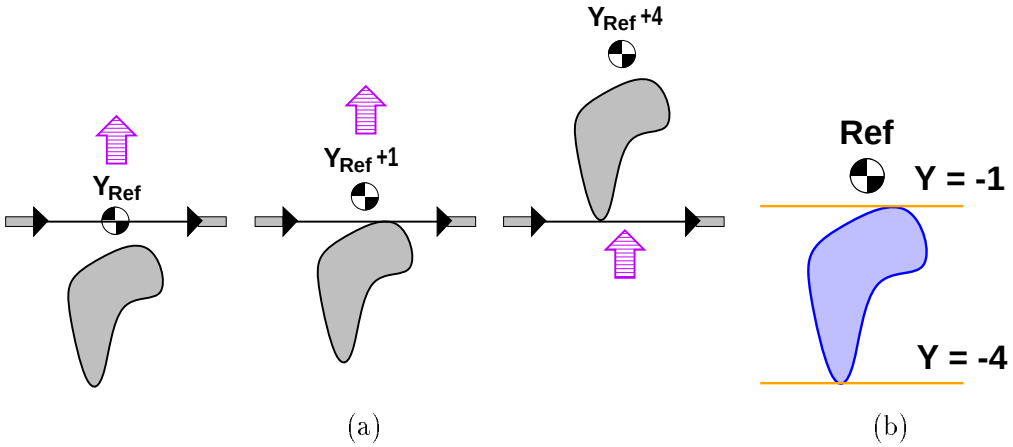


Figure 2.5: The relative break points are defined with respect to the *virtual* break point Y_{Ref} which refers to the theoretical breakpoint corresponding to the origin of the reference frame.

Each light beam sensor produces two break points, one for occlusion and one for when the beam reconnects, providing two line probe constraints. A crossbeam sensor's b beam sensors provide $2b$ line probe constraints, which characterize an enclosing parallelepiped that must contact and contain the object (Figure 2.6). The problem of identifying an object from an enclosing parallelepiped has more structure than the general problem of identifying an object from an enclosing convex polygon which has been examined by Cole and Yap [CY87] because, for the crossbeam sensor, the parallelepiped edge/line probe orientations are fixed.

In this chapter, we are focusing on the correspondence problem, the task of interpreting the sensed features as model features, which, in this case, is the task of identifying model features which correspond to the parallelepiped edges. The object's (x, y) translation

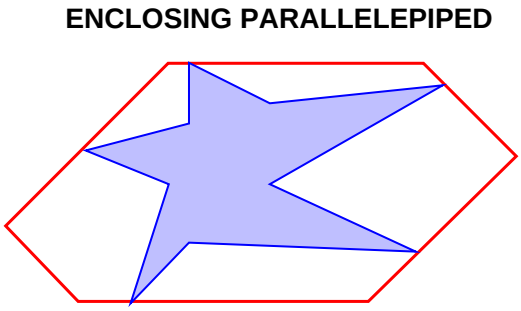


Figure 2.6: The combination of constraints from many beam sensors form an enclosing parallelepiped which must contact and contain the object.

is irrelevant for the purpose of solving the correspondence problem because parallelepiped geometry (Figure 2.7) depends only upon the orientation θ. Thereby, we are dealing with a relatively simple one degree of freedom problem.

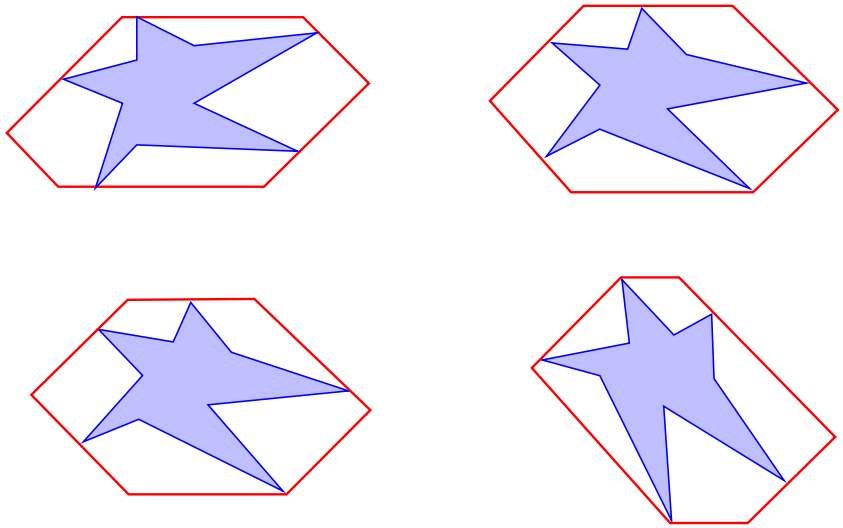


Figure 2.7: The geometry of the enclosing parallelepiped is independent of x and y and depends only upon the orientation θ.

2.2.1 Localization

For each candidate interpretation of model features (vertices) as sensed features (parallelepiped edges), we can compute the optimal pose estimate via a least squares error approach. Objects are localized by finding the transformation which best maps the model

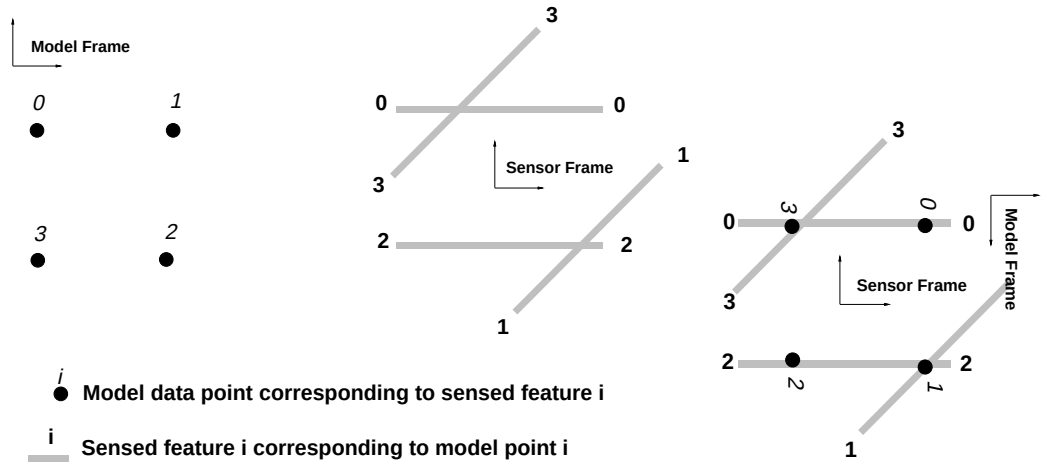


Figure 2.8: (a) A set of model data points in the model reference frame, and sensed line features in a sensor reference frame, (b) the task is to determine the position of the model in the sensed reference frame by finding the transformation which optimally maps the model points onto the sensed features.

vertices onto the corresponding parallelepiped edges. For circular arc features, we can reduce the problem of matching a circular arc to a parallelepiped edge to the problem of matching a point (the center of the circular arc) to the parallelepiped edge translated inwards by the radius of the circular arc.

Assuming perfect modeling and perfect sensing, the object's pose would correspond to the two-dimensional rigid transformation which transforms the model vertices exactly onto the associated parallelepiped edges. Real sensors and imperfect models contribute small errors, producing inconsistent, overconstrained situations; least squares approaches are common methods for solving these types of overconstrained problems.

We formulate pose estimation as a least squares problem which involves finding the two-dimensional rigid transformation which minimizes the sum squared distances between the transformed vertices and associated parallelepiped edges. The advantage of least squared methods is that they provide the maximum a posterior estimate assuming a normal error distribution. Furthermore, least squares approaches can accommodate non-uniform sensor precisions by suitably scaling the errors attributable to each sensor. We do not scrutinize this aspect of crossbeam sensing because many algorithms for solving least squares problems are found in the literature. We used the technique described by Wallack and Manocha [WM94a] which is a constant time, exact algorithm for estimating the transformation which maps

vertices onto associated linear features (Figure 2.8).

2.3 Theoretical Framework

In this section, we provide the theoretical framework for both the linear-time online correspondence algorithm, and the constant-time online correspondence algorithm which utilizes preprocessing.

2.3.1 Notation

Throughout this chapter, caligraphic notation will signify predicted values, and bold faced terms will signify experimentally measured values; *i.e.*, $\mathcal{D}$ refers to a predicted measurement, whereas $\mathbb{D}$ refers to an experimental measurement.

2.3.2 Extracting Maximal Information from Break Point Data

The problem of identifying objects from break point data reduces to the problem of finding a model which exactly fits inside a given parallelepiped. A $2b$ sided parallelepiped with prespecified edge orientations only has $2b \Leftrightarrow 2$ degrees of freedom. Two degrees of freedom are lost because the edges must form a closed loop; in other words, if we choose the lengths of the first $2b \Leftrightarrow 2$ edges, the lengths of the remaining two edges are constrained by the fact that the endpoint of the last edge coincides with the starting point of the first edge. We parameterize parallelepipeds in terms of b diameters and $b \Leftrightarrow 2$ inset distances, which we will define shortly (Figure 2.9). Since this parameterization provides $2b \Leftrightarrow 2$ independent measurements and the system only has $2b \Leftrightarrow 2$ degrees of freedom, this parameterization is not rank-deficient.

2.3.3 Diameter $\mathcal{D}_f$

Definition 2.1 *Diameter* $\mathcal{D}_f$ of an object f (Figure 2.10) refers to the width of the projection of object f onto the y-axis (the measurement axis is arbitrary). The diameter function $\mathcal{D}_f(\theta)$ refers to the predicted diameter when f is rotated counterclockwise by θ. $\mathbb{D}$ refers to a measured diameter, and $\mathbb{D}^j$ refers to the diameter measured normal to beam B_j. $\mathcal{D}_f^j(\theta)$ signifies the diameter predicted for beam sensor j, which corresponds to shifting the diameter function by the beam's orientation: $\mathcal{D}_f^j(\theta) = \mathcal{D}_f(\theta \Leftrightarrow \angle B_j)$. $\mathcal{D}_f^{-1}(\mathbb{D} \pm \epsilon_{\mathcal{D}})$ is defined as the

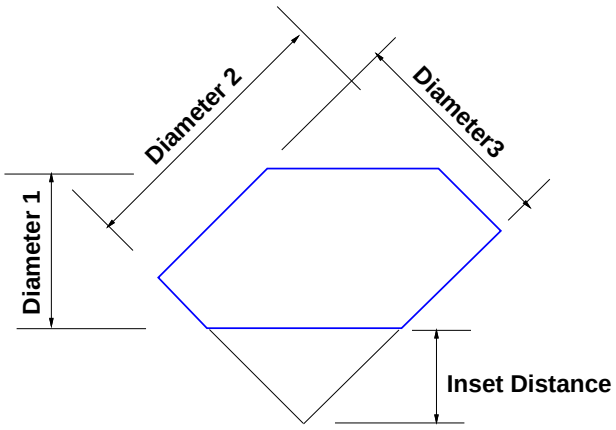


Figure 2.9: Parallelepipeds are parameterized by measured diameters and inset distances. A $2b$ sided parallelepiped provides b diameters and $b \Leftrightarrow 2$ inset distances. In this case, $b = 3$, and the six-sided parallelepiped provides 3 diameters and 1 inset distance.

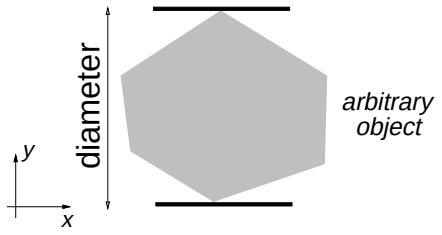


Figure 2.10: The diameter $\mathcal{D}$ of a polygon.

set of orientations with diameters consistent with $\mathbb{D} \pm \epsilon_{\mathcal{D}}$. For shorthand, the $_f$ subscript may be omitted since we will deal with one object at a time.

The diameter corresponds to the maximum y difference between two points in the object. Locally, the diameter function depends upon the two features with extremal y coordinates, as shown in Figure 2.11. Overall, the diameter function $\mathcal{D}_f(\theta)$ is composed of sections $\mathcal{D}_{L_i}$ of positively offset sinusoidal arcs of the form $(\sin(\theta+\alpha)+b \mid \alpha, b \in \mathbb{R}; b > 0)$ which correspond to vectors L_i between pairs of extremal features (vertices or the centers of circular arcs). $\angle L_i$ refers to the orientation of the vector L_i, and $|L_i|$ refers to the length of the vector L_i.

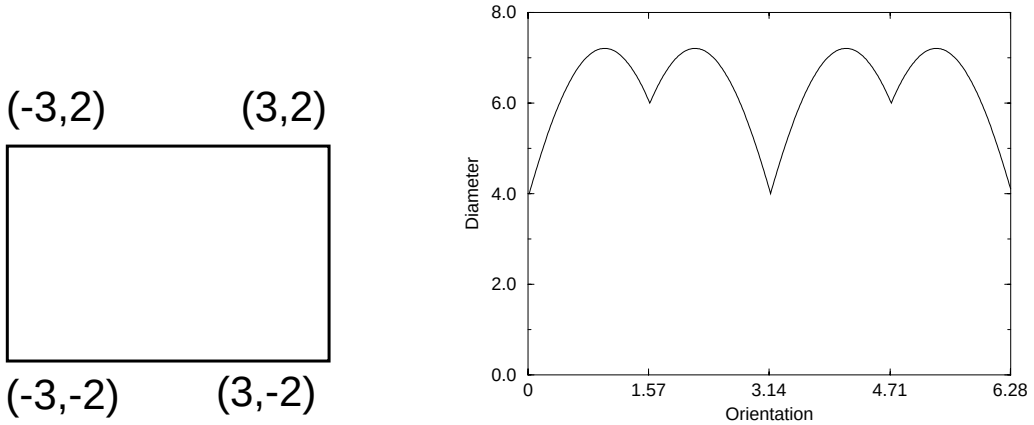


Figure 2.11: Diameter function of a 2:3 rectangle.

For polygonal objects, when the extremal features are vertices, the diameter function can be expressed by equation (2.1). For generalized polygonal objects, or when one of the extremal features is a circular arc, the vector L_i corresponds to the centers of the circular arcs (Figure 2.13). In this case, the total diameter function is the sum of the constant component due to the radii of the circular arcs, $r_{\Sigma_i} = r_{i,1} + r_{i,2}$, and the sinusoidal component due to the vector L_i, $|L_i| \sin(\theta + \angle L_i)$ (equation (2.2)).

$$\mathcal{D}_{L_i}(\theta) = |L_i| \sin(\theta + \angle L_i) \tag{2.1}$$

$$\mathcal{D}_{L_i}(\theta) = \underbrace{r_{i,1} + r_{i,2}}_{r_{\Sigma_i}} + |L_i| \sin(\theta + \angle L_i) \tag{2.2}$$

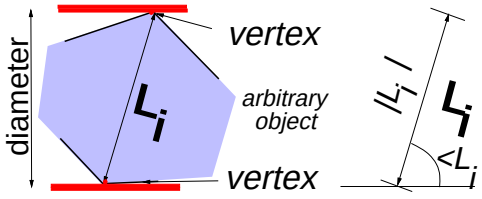


Figure 2.12: The vector L_i between two vertices.

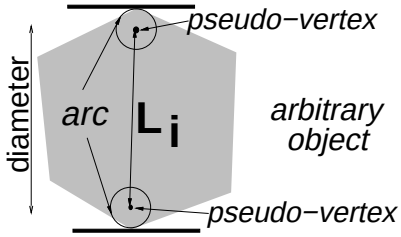


Figure 2.13: The vector L_i between two pseudo vertices corresponding to the centers of circular arcs.

Inverse Diameter Function $\mathcal{D}^{-1}$

$\mathcal{D}^{-1}(\mathbb{D} \pm \epsilon_{\mathcal{D}})$ refers to the orientation ranges consistent with measured diameter $\mathbb{D} \pm \epsilon_{\mathcal{D}}$, where $\epsilon_{\mathcal{D}}$ refers to a conservative error estimate. These orientation ranges are computed by scanning through all of the sinusoidal diameter function components $\mathcal{D}_{L_i}$, and computing the orientation ranges consistent with $\mathbb{D} \pm \epsilon_{\mathcal{D}}$ (equation (2.3)).

$$\mathcal{D}^{-1}(\Delta) = \bigvee_{L_i} \left(\pm \arcsin\left(\frac{\mathbb{D} \pm \epsilon_{\mathcal{D}} \Leftrightarrow r_{\Sigma_i}}{|L_i|}\right) \Leftrightarrow \angle L_i \right) \tag{2.3}$$

The orientation ranges consistent with a measured diameter $\mathbb{D} \pm \epsilon_{\mathcal{D}}$ is computable in $O(n)$ time, where n is the number of object features, via brute force enumeration, but they can also be computed in $O(A + \log(n))$ time, where A refers to the number of ranges returned by treating it as an interval overlap problem, or in $O(A)$ time by caching the orientation ranges consistent with all possible discretized diameters $\mathcal{D}^{-1}(\lfloor \mathbb{D} \rfloor)$.

2.3.4 Inset Distance

Definition 2.2 *Inset distance* $\mathcal{I}$, corresponding to a triplet of edges, refers to the distance, measured normally, from first edge to the intersection point of the other two edges (Figure 2.15). $\mathbb{I}^{w,y,z}$ refers to the measured inset distance corresponding to base beam B_w, and

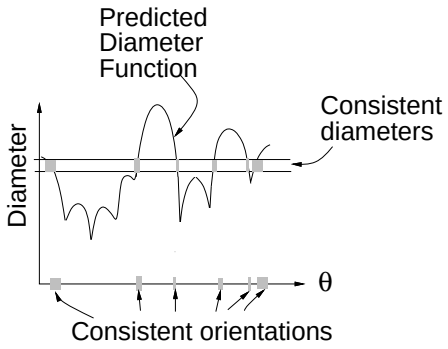


Figure 2.14: $\mathcal{D}^{-1}(\mathcal{D} \pm \epsilon_{\mathcal{D}})$ characterizes the set of possible orientations consistent with the sensed diameter.

side beams B_y, B_z. $\mathcal{I}^{w,y,z}$ refers to the predicted inset distance corresponding to base beam B_w, and side beams B_y, B_z. We use the shorthand terms $\mathbb{I}^k$ and $\mathcal{I}^k$ to denote $\mathbb{I}^{w,y,z}$ and $\mathcal{I}^{w,y,z}$.

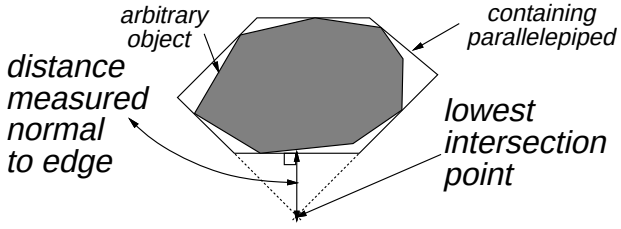


Figure 2.15: The inset distance corresponding to three edges.

Equation (2.4) expresses $\mathcal{I}^k(\theta)$ as a function of the orientation θ. In Figure 2.16, the inset distance is measured normally to the bottom edge w; V_{AB} and V_{AC} refer to the vectors connecting the bottom contact vertex A to the other vertices B, C on edges Y and Z. For a particular triplet of beams, there are $3n$ different triplets of for an n sided object. This is shown by observing that if we walk a tangent line around the object's boundary, the contact point changes n times. Therefore, three relatively oriented tangent lines generically change contact vertices $3n$ times.

$$\begin{aligned} \beta &= \angle B_y \Leftrightarrow \angle B_w \\ \gamma &= \angle B_z \Leftrightarrow \angle B_w \\ \mathcal{I}_f^{w,y,z}(\theta) &= \frac{1}{\sin(\beta+\gamma)}(G_{AB}(\theta)\sin(\gamma) + G_{AC}(\theta)\sin(\beta)) \end{aligned} \tag{2.4}$$

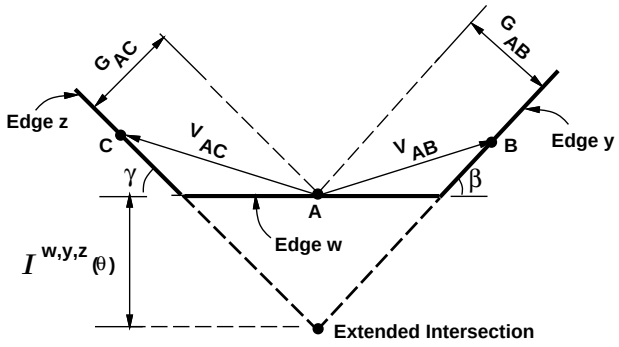


Figure 2.16: $\mathcal{I}^k(\theta)$ is shorthand for $\mathcal{I}^{w,y,z}(\theta)$.

$$= \frac{\Leftrightarrow 1}{\sin(\beta+\gamma)}(|V_{AB}|\sin(\angle V_{AB}+\theta \Leftrightarrow \beta)\sin(\gamma)+|V_{AC}|\sin(\angle V_{AC}+\theta \Leftrightarrow \gamma)\sin(\beta))$$

The inset distance function can also be extended to handle generalized polygons as shown in Figure 2.17. Equation (2.5) expresses the inset distance for circular arc features, and equation (2.6) expresses a generic formulation for the predicted inset distance function.

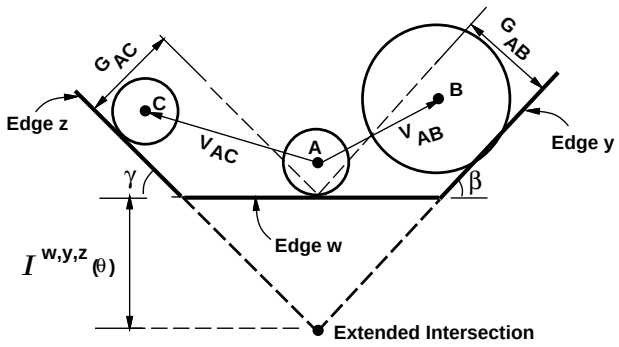


Figure 2.17: The $\mathcal{I}^{w,y,z}(\theta)$ model for generalized polygons.

$$\mathcal{I}_f^{w,y,z}(\theta) = \frac{\Leftrightarrow 1}{\sin(\beta+\gamma)}((|V_{AB}|\sin(\angle V_{AB}+\theta \Leftrightarrow \beta) \Leftrightarrow r_A\cos(\alpha)+r_B)\sin(\gamma)+ (|V_{AC}|\sin(\angle V_{AC}+\theta \Leftrightarrow \gamma) \Leftrightarrow r_A\cos(\alpha)+r_C)\sin(\beta)) \quad (2.5)$$

$$= Q_1\sin(\theta+Q_2)+Q_3 \quad (2.6)$$

Inverse Inset Distance Function $\mathcal{I}^{-1}$

$\mathcal{I}^{-1}(\mathbb{I}\pm\epsilon_{\mathcal{I}})$ refers to the orientation ranges consistent with measured inset distance $\mathbb{I}\pm\epsilon_{\mathcal{I}}$, where $\epsilon_{\mathcal{I}}$ refers to a conservative error estimate. These orientation ranges are com-

puted by scanning through all of the vertex triplets A, B, C, and computing the orientation ranges consistent with $\mathbb{I} \pm \epsilon_{\mathcal{I}}$ (equation (2.3)).

$$\mathcal{I}^{-1}(\mathcal{I}) = \bigvee_{A,B,C} \left(\pm \arcsin\left(\frac{\mathbb{I} \pm \epsilon_{\mathcal{I}} \Leftrightarrow Q_3}{Q_1}\right) \Leftrightarrow Q_2 \right) \tag{2.7}$$

2.4 Online Correspondence Algorithm

In this section, we describe a linear time online algorithm for solving the correspondence problem. This algorithm takes as input the set of candidate models $F = \{f_1, f_2, \ldots, f_m\}$ and the enclosing parallelepiped measured by the crossbeam sensor. The online correspondence algorithm running time is linear in the number of beams b, the number of objects $|F|$, and the number of features in each object model $n = \max_i |f_i|$. For each object model, we enumerate ranges of orientations with predicted diameters consistent with the sensed diameters for each beam. Then we intersect these sets of orientation ranges, and then verify the diametrically-consistent orientation ranges by comparing the predicted inset distances with the measured inset distances.

1. For each cross section $f \in F$:

   (a) Compute $\Theta^j = \mathcal{D}_f^{j,-1}(\mathbb{D}^j \pm \epsilon_{\mathcal{D}})$ for each beam j by computing $\mathcal{D}_f^{-1}(\mathbb{D}^j \pm \epsilon_{\mathcal{D}})$ and then shifting the orientation ranges by $\angle B_j$ (Figure 2.14, equations (2.1), (2.2)).

   (b) Compute Θ^T the orientations consistent with all the sensed data by intersecting the orientation ranges consistent with each sensed diameter: $\Theta^T = \bigcap_{j=1,\ldots,b} \Theta^j$, by scanning down θ.

   (c) Validate the orientation ranges of Θ^T by comparing the measured inset distances $\mathbb{I}^k$ with the predicted inset distance $\mathcal{I}_f^k(\theta)$ ($|\mathbb{I}^k \Leftrightarrow \mathcal{I}_f^k(\theta)| < \epsilon_{\mathcal{I}}$).

The algorithm's complexity is $O(|F|bn)$, where $|F|$ refers to the number of candidate cross-section models, b refers to the number of beams, and n refers to the number of features in each model. For each of the $|F|$ object cross-section models, constructing the diameter function takes $O(n)$ time, computing $\mathcal{D}_f^{-1}(\mathbb{D} \pm \epsilon_{\mathcal{D}})$ takes $O(n)$ time for each beam, and intersecting of the ordered orientation ranges for b beams by scanning down θ takes $O(bn)$ time.

2.5 Localizing Objects In Constant Time via Indexing

In this section, we describe a constant-time indexing approach for solving the correspondence problem for crossbeam sensor data. Indexing involves extracting coordinates from the crossbeam sensor data, quantizing the indexing coordinates, and using the discretized coordinates to index a table entry containing the valid interpretations (Figure 2.18). The crossbeam sensor perceives a parallelepiped, and indexes using $2b \Leftrightarrow 2$ coordinates $\mathbb{X}$ including b measured diameters $\mathbb{D}^1, \mathbb{D}^2, \ldots \mathbb{D}^b$, and $b \Leftrightarrow 2$ measured inset distances $\mathbb{I}^1, \mathbb{I}^2, \ldots \mathbb{I}^{b-2}$. Indexing relies upon the assumption that the measured sensor data will be discretized into the same table entry as the predicted sensor data. This is a reasonable assumption because we can choose the discretization resolution to accommodate sensor and modeling error.

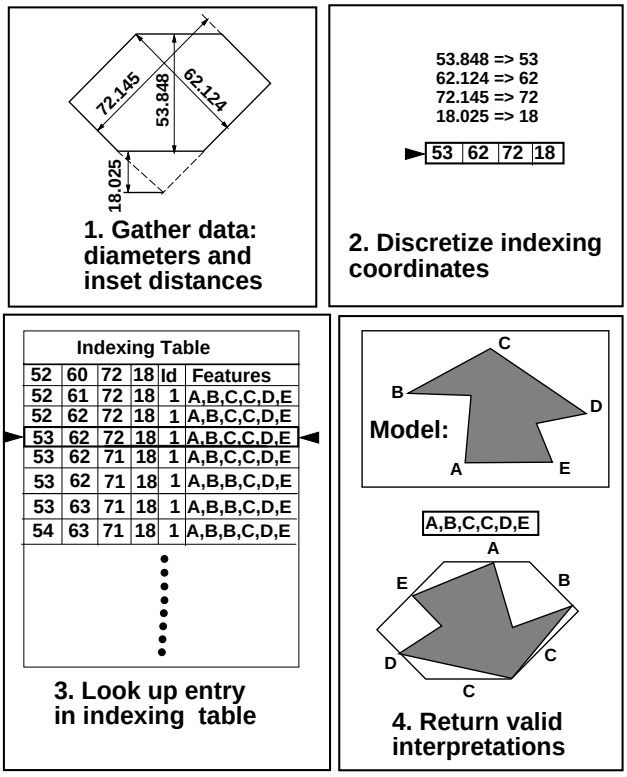


Figure 2.18: Indexing techniques involve discretizing the sensed data to index a table entry containing valid interpretations of the sensed features.

Definition 2.3 *Indexing coordinates* $\mathcal{X}_f(\theta)$ refer to the predicted parallelepiped parameters $\mathcal{D}_f^1(\theta), \mathcal{D}_f^2(\theta), \ldots \mathcal{D}_f^b(\theta), \mathcal{I}_f^1(\theta), \mathcal{I}_f^2(\theta), \ldots \mathcal{I}_f^{b-2}(\theta)$, and $\mathbb{X}$ refers to the experimentally measured parallelepiped parameters: $\mathbb{D}^1, \mathbb{D}^2, \ldots \mathbb{D}^b, \mathbb{I}^1, \mathbb{I}^2, \ldots \mathbb{I}^{b-2}$. In order to achieve in-

tegral indices to perform hash table lookups, the indexing coordinates are quantized by resolution λ: $\mathcal{X}_f(\theta) \rightarrow \lfloor \frac{\mathcal{X}_f(\theta)}{\lambda} \rfloor$, $\mathbb{X} \rightarrow \lfloor \frac{\mathbb{X}}{\lambda} \rfloor$.

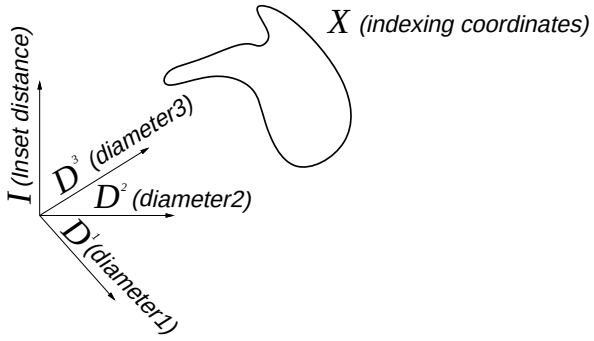


Figure 2.19: Since the parallelepiped parameters depend only upon θ, the predicted indexing coordinates trace out a one dimensional curve in a $2b \Leftrightarrow 2$ dimensional indexing space: $\mathcal{X}(\theta) = \mathcal{D}^1(\theta), \mathcal{D}^2(\theta), \mathcal{D}^3(\theta), \mathcal{I}(\theta)$.

Definition 2.4 *Correspondence interpretation* $\mathcal{C}_f(\theta)$ refers to the predicted model vertices contacting the parallelepiped edges after rotating the object counterclockwise by θ.

Definition 2.5 The *Indexing table* $\mathcal{T}_f$ provides a mapping from the predicted parallelepiped measurements $\mathcal{X}_f(\theta)$ to the correspondence interpretation $\mathcal{C}_f(\theta)$ by including entries for valid combinations of indexing coordinates and correspondence interpretations $\langle \mathcal{X}_f, \mathcal{C}_f \rangle$. The indexing table provides the desired map from $\mathbb{X}$ to $\mathbb{C}$ assuming the predicted measurements and the actual measurements are discretized to the same quantized coordinates.

In order to use indexing to solve the correspondence problem for crossbeam sensing, it is critical that the indexing tables be complete. Constructing a complete lookup table is not a trivial task. If configuration space was finite, a simple indexing table construction algorithm would involve predicting indexing table entries for each configuration: $\mathcal{T}_f(\mathcal{X}_f(\theta_1), \mathcal{C}_f(\theta_1))$, $\mathcal{T}_f(\mathcal{X}_f(\theta_2), \mathcal{C}_f(\theta_2)), \ldots, \mathcal{T}_f(\mathcal{X}_f(\theta_N), \mathcal{C}_f(\theta_N))$, but this approach is inadequate for non-finite configuration spaces, such as those with continuous degrees of freedom. Another approach involves predicting table entries $\mathcal{T}_f(\mathcal{X}_f(0), \mathcal{C}_f(0))$, $\mathcal{T}_f(\mathcal{X}_f(\theta), \mathcal{C}_f(\theta))$, $\mathcal{T}_f(\mathcal{X}_f(2\theta), \mathcal{C}_f(2\theta))$, ... $\mathcal{T}_f(\mathcal{X}_f(k\theta), \mathcal{C}_f(k\theta))$ for regularly sampled configurations $k\theta | k \in \mathbb{Z}$, or randomly sampled configurations, but this approach fails to guarantee completeness.

2.5.1 Complete Indexing Tables

In this section, we describe a provably complete approach for constructing lookup tables for any system characterizable by a single degree of freedom. The basic idea of this technique is that we walk along the predicted indexing coordinates and enumerate table entries whenever we change discretized indexing coordinates or correspondence interpretations. For crossbeam sensing, we walk along θ (Figure 2.19) making adaptively sized steps so that we do not skip over any indexing table entries. We guarantee not to skip over any entries by determining the critical orientations for which either the discretized coordinates or correspondence interpretation change. To compute the critical orientations where the discretized indexing coordinates change, we use the inverse diameter function and inverse inset distance function to compute $theta_{k\lambda} = \mathcal{X}^{-}1(k\lambda)|k \in \mathbb{Z}$. We critical orientations for interpretation changes by predicting the orientations for which the features contacting the enclosing parallelepiped contacts change. Thereby, we can enumerate every table entry $\mathcal{T}_f(\mathcal{X}_{f,1},\mathcal{C}_{f,1}), \mathcal{T}_f(\mathcal{X}_{f,2},\mathcal{C}_{f,1}), \ldots$ consistent with an object model.

Complete indexing tables also allow us to predict how the probability that the crossbeam sensor can identify objects uniquely. We can use the complete indexing tables to identify configurations for which the crossbeam technique may not be able to distinguish between two objects in different configurations by looking for indexing table entries with the same discretized coordinates. This heuristic can be used to estimate the probability that a single sensing operation will fail to uniquely identify an object.

The two main caveats of this approach are that the technique assumes that the object moves through the crossbeam sensor in a constant orientation, and and that each different crossbeam sensor may require different indexing tables for all of the workpieces.

2.5.2 Sizes of Complete Indexing Table for Various Objects

Table 2.1 shows the sizes of indexing tables for various objects (Figures 2.8,2.26) and discretization resolutions. The indexing tables are constructed by starting with $\theta = 0$ and increasing θ and predicting indexing table entries $\mathcal{T}(\mathcal{X}(\theta),\mathcal{C}(\theta))$ until $\theta > \frac{2\pi}{symmetry}$. Entries labeled with an asterisk denote a generalized polygonal model, and those without asktersks denote polygonal models. From Table 2.1, we can observe that expressive generalized polygonal models shrink the sizes of the indexing tables, but that the effect becomes less pronounced as the discretization becomes finer.

The indexing table construction algorithm runs in $O(A)$ time, where A is the number of indexing table entries (Table 2.1). Each table entry includes $2b \Leftrightarrow 2$ integer values and $2b$ feature descriptors. For a three beam crossbeam sensor, each table entry takes up 40 bytes, and each complete indexing table takes up a few tens of kilobytes.

Object	Table Size for $\delta = 1$cm	Table Size for $\delta = 0.1$cm
legoUShape	370	2374
yellowWhistle*	258	1468
yellowWhistle	666	3395
markerCap	203	557
penCap*	422	3397
penCap	543	3515
twentyPin	580	3890
legoTriangle	319	2140
quarterCircle	418	2289
legoSquare	57	479
legoL	247	1436
legoRectangle	176	1681
keyence	530	3738

Table 2.1: Sizes of indexing tables for various objects at two different resolutions. Most objects were modeled with polygonal models, and asterisks (*) denote generalized polygonal models.

2.5.3 Sensor and Object Error

In order to satisfy performance requirements, the indexing technique must be robust to sensor noise and modeling error (which can be defined in terms of error tolerances $\epsilon_{\mathcal{D}}, \epsilon_{\mathcal{I}}$). In order to tolerate these errors, we must consider all of the indexing coordinates within an $\langle \epsilon_{\mathcal{D}}, \epsilon_{\mathcal{I}} \rangle$ neighborhood. We can search this neighborhood in three different ways: (a) *fuzzing* the indexing table by including extra table entries for the $< \epsilon_{\mathcal{D}}, \epsilon_{\mathcal{I}} >$ neighborhood surrounding predicted coordinates, (b) searching the neighborhood surrounding the measured indexing coordinates at runtime, or (c) utilizing specialized data structures such as a fat space partition [Ove92], which uses a reasonably sized data structure and performs table lookups in constant time. The first approach has the drawback that it significantly increases the number of table entries, possibly making the table too large to reside in main memory, which may cause thrashing. We choose the second approach because brute force enumeration runs in reasonable time; with four coordinate axes, searching over an error

range of ± 1 on each coordinate axis amounts to $3^{2b-2} = 3^4 = 81$ table lookups. Each indexing table lookup operation takes 5 microseconds on a SPARC 1, and performing 81 lookups only takes 0.4 milliseconds. We did not choose to use a specialized data structure because the brute force approach was satisfactory.

2.6 Crossbeam Sensor Calibration

Accurate beam position information is one of the most critical prerequisites for crossbeam sensing. In this section, we describe a calibration technique for localizing the beams [Wal95c] process which can involve as few as five measurements. In this section, we discuss the non-ideal characteristics of beam sensors and present experimental results which validate our beam sensor model. Sensor calibration requires an accurate x, y positioner, such as a micrometer, or a Robotworld system. In order to normalize out any dependence on beam orientation, we use a cylindrical peg so that all the beam sensors perceive the same cross-section regardless of beam orientation.

2.6.1 Localizing the Beams

In section 2.2 we explained the rationale for using relative break points rather than the original break points. Relative break points were defined with respect to the reference frame. To simplify the calibration process, we choose the center of rotation of the calibrating peg as the center of the reference frame.

The beams are localized by attaching a cylindrical calibration peg to the end-effector, and then passing the calibration peg along different paths through the crossbeam sensor, as shown in Figure 2.20. The beam positions are estimated by lines which best fit the measured break points shown in Figure 2.21.

As few as five experiments should provide sufficient precision although the precision improves with the sample size. Performing a large number of trials results in higher precision under the assumption that the variation between measured break points and the predicted break points can be characterized by an independent random variable. In this case, the standard deviation of the beam position estimates drop off as $\frac{1}{\sqrt{N}}$.

We attempted to tailor the calibration process to normalize out the effects of the non-ideal characteristics of the beam sensors. We handled peg asymmetry by passing the calibration peg through the crossbeam sensor at diametrically opposite orientations, and

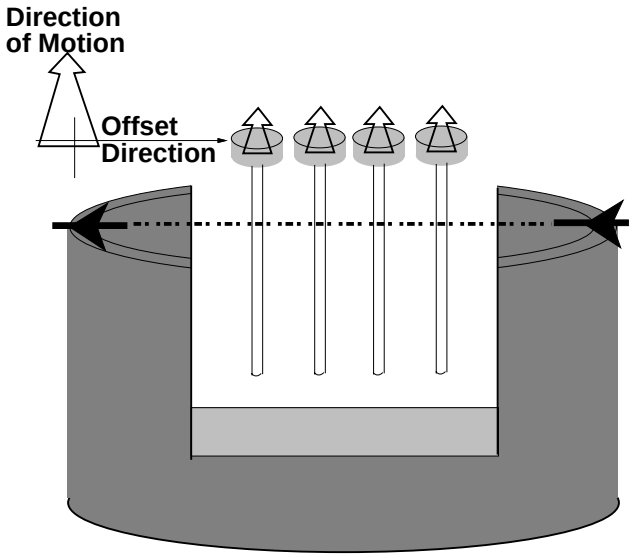


Figure 2.20: The beams are localized by passing a $\frac{1}{4}''$ diameter calibrating peg through the crossbeam sensor along various paths at diametrically opposite orientations. In this crossbeam sensor, the beam sensors are rigidly attached to a cylindrical chassis so that all of the light beams span the same distance.

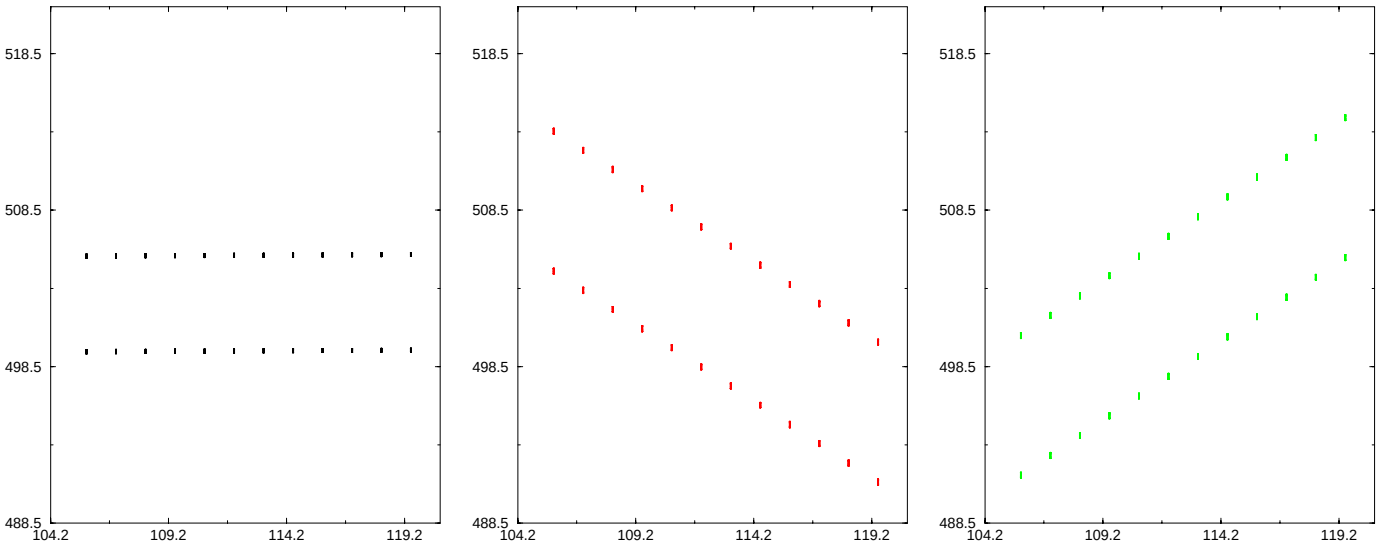


Figure 2.21: Calibration break point measurements.

we handled beam hysteresis by averaging the break points corresponding to backwards and forwards travel through the crossbeam sensor.

2.6.2 Non-Ideal Characteristics of Light Beam Sensors

Robustness, low cost, and accuracy make light beam sensors suitable for industrial applications. The main advantages of light beam sensing are non-contact, lack of moving parts, and the sensors' simplicity. The sensors' high precision repeatability and high speed are attributable to the sensors' simple specification.

For prismatic objects, the non-ideal characteristics only slightly affect the break point measurements, on the order of a few tens of microns. An ideal light beam would be have zero width, and have zero latency between the time the beam becomes occluded or reconnects and the sensor output changes. Real beam sensors exhibit non-ideal characteristics because real beams are not infinitesmally thin, because the opto-electronics are based on charge collection capacitors, and because the sensing circuitry uses thresholding to stabilize their output. The time delays associated with the opto-electronic devices can be minimized by increasing the luminosity of the beam source and correspondingly cooling the detector.

The non-infinitesmal beam width results in a narrow range of differently oriented beams. Consequently, curvature in the z direction and specularity affect the accuracy of the break point measurements. We used commercially available beam sensors with beam apertures of a few millimeters. In this chapter, we focused on objects whose cross-sections remained relatively constant around the beam plane. More precise beam sensors would enable recognizing objects where the cross-sections varied more strongly. Furthermore, our beam sensors also exhibited crosstalk because two were modulated at the same frequency. The performance of crossbeam sensors would be markedly improved by using beam sensors with increased the beam span, which effectively tightens the aperture and shrinks the beam columns down to a line, and using beam sensors which did not exhibit crosstalk and interreflection by synchronously pulsing the beams so that only one light source is active at a time.

Deviations from Expected Values

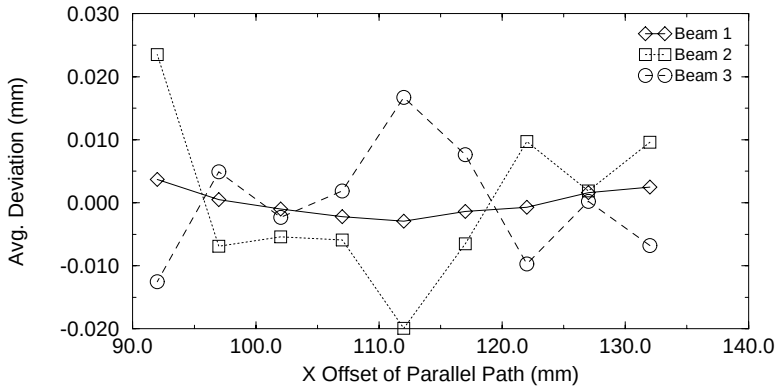


Figure 2.22: The average discrepancy between the actual break point measurements and predicted break point measurements.

We experimentally validated our linear model of light beam sensors and determined that the model was accurate to an accuracy of 50 microns. The experiment involved precisely estimating the beam positions using a large data set of calibration measurements, and then comparing the measured break points with the predicted break points (using the linear model) to highlight systematic discrepancies. The beams were localized by passing the calibration peg along nine parallel paths, 144 times along each path. Along the extremal paths, the calibration peg passed nearby the light emitter and receiver devices. We observed a systematic discrepancy between the predicted break points and the actual break points which was noticeably dependent upon the path and and the beam orientation.

Figure 2.22 shows that for the $\pm 45^o$ beams, the variation between the actual and expected break points along the middle paths differed from the variation between the actual and expected break points along the extremal paths by approximately 50 microns. For the horizontal beams, the difference was markedly less.

2.7 Example

We now run through an example to illustrate how objects are recognized and localized for crossbeam sensors using the indexing approach: first we extract indexing coordinates from the parallelepiped geometry, and then we look up the correspondence information in the indexing table. After solving the correspondence problem, we focus on the localization problem, and solve it by a reduction to a nonlinear least squares problem. In this example, the object was a whistle shown in Figure 2.23.

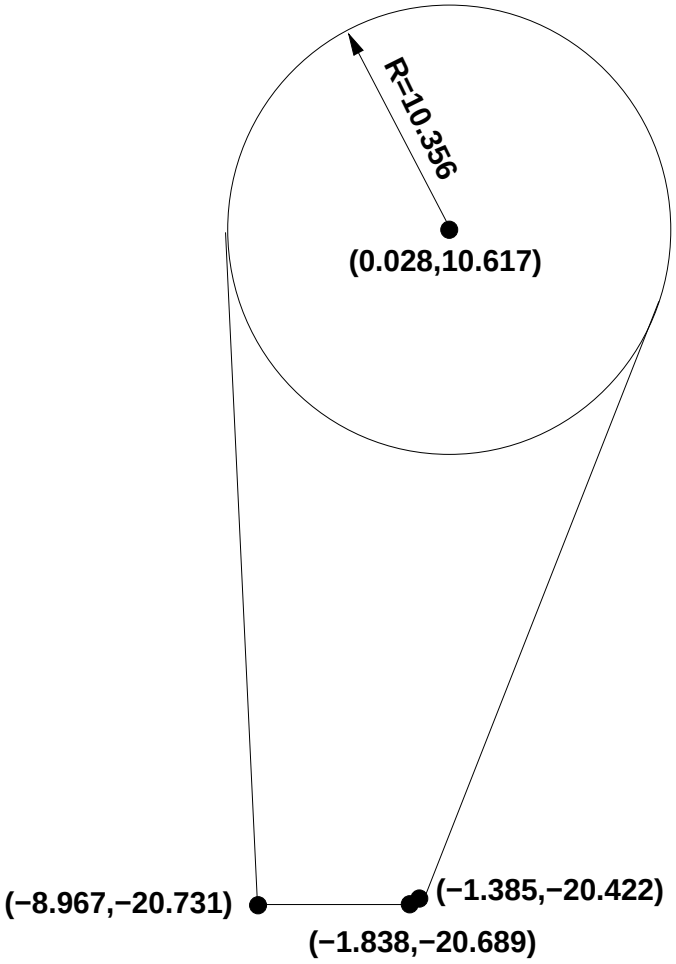


Figure 2.23: The relevant features of the whistle object used in this example.

Beam	Estimated Line
1	$y = \tan(0.006151\,\text{rad})x + 501.7850$
2	$y = \tan(\Leftrightarrow 0.777978\,\text{rad})x + 613.3710$
3	$y = \tan(0.791683\,\text{rad})x + 388.7994$

Table 2.2: The estimated lines corresponding to the light beam sensors.

Beam	First Break Point	Second Break Point
1	(111.9991, 479.1447)	(111.9991, 520.4242)
2	(111.9991, 468.5893)	(111.9991, 525.2973)
3	(111.9991, 482.7245)	(111.9991, 524.7521)

Table 2.3: The break points obtained by passing the whistle through the crossbeam sensor along the path $x = 111.9991$.

The estimated beam positions are given in Table 2.2, and the break points resulting from passing the square through the crossbeam sensor are given in Table 2.3. First we compute the relative break points, relative to the *virtual* break point of the end effector reference frame.

Figure 2.24 depicts the enclosing parallelepiped defined by the relative break point constraints. The three diameters $\{\mathbb{D}^1, \mathbb{D}^2, \mathbb{D}^3\}$ are computed to be $\{41.2787, 40.3951, 29.5306\}$.

The inset distance is measured by finding the intersection of the bottom left and bottom right sides of the parallelepiped $(\Leftrightarrow 7.4787, \Leftrightarrow 27.0641)$, and measuring that normal to the bottom edge yields a contact distance of 3.6888.

The indexing coordinates $\mathbb{X} = [41.2787, 40.3951, 29.5306, 3.6888]$, and discretized $\lfloor \frac{\mathbb{X}}{\lambda} \rfloor$. Then we look up the corresponding entry $(41, 40, 29, 3)$ in the indexing table. The indexing table entry specifies the model features corresponding to the sensed features.

2.7.1 Pose Estimation

The localization algorithm described by Wallack and Manocha [WM94a] estimates the optimal pose which transforms the model vertices onto the parallelepiped edges with minimum total squared error, to be: ($X = 0.842662$, $Y = 2.913285$ $\theta = 1.492074$ radians) as shown in Figure 2.25.

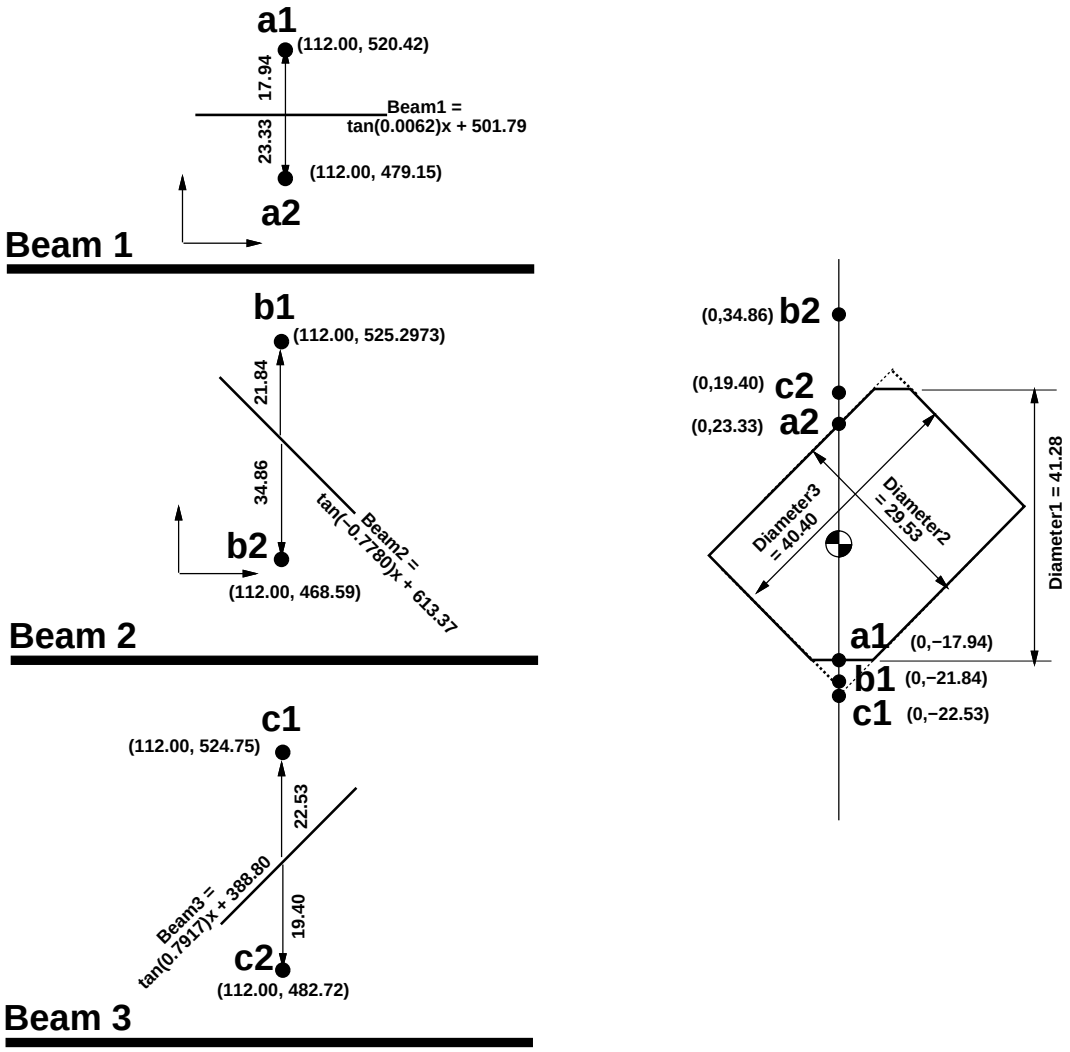


Figure 2.24: The enclosing parallelepiped defined by the relative break point constraints. The six break points $a_1, a_2, b_1, b_2, c_1, c_2$ define the edges of the parallelepiped.

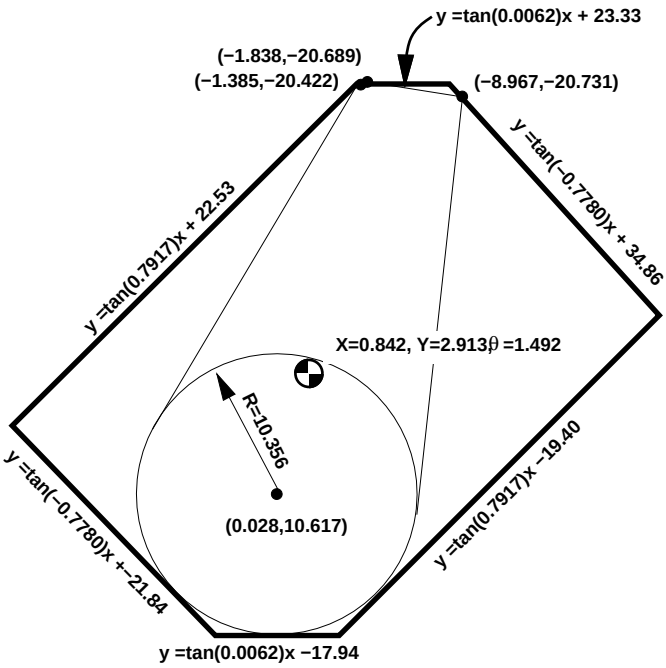
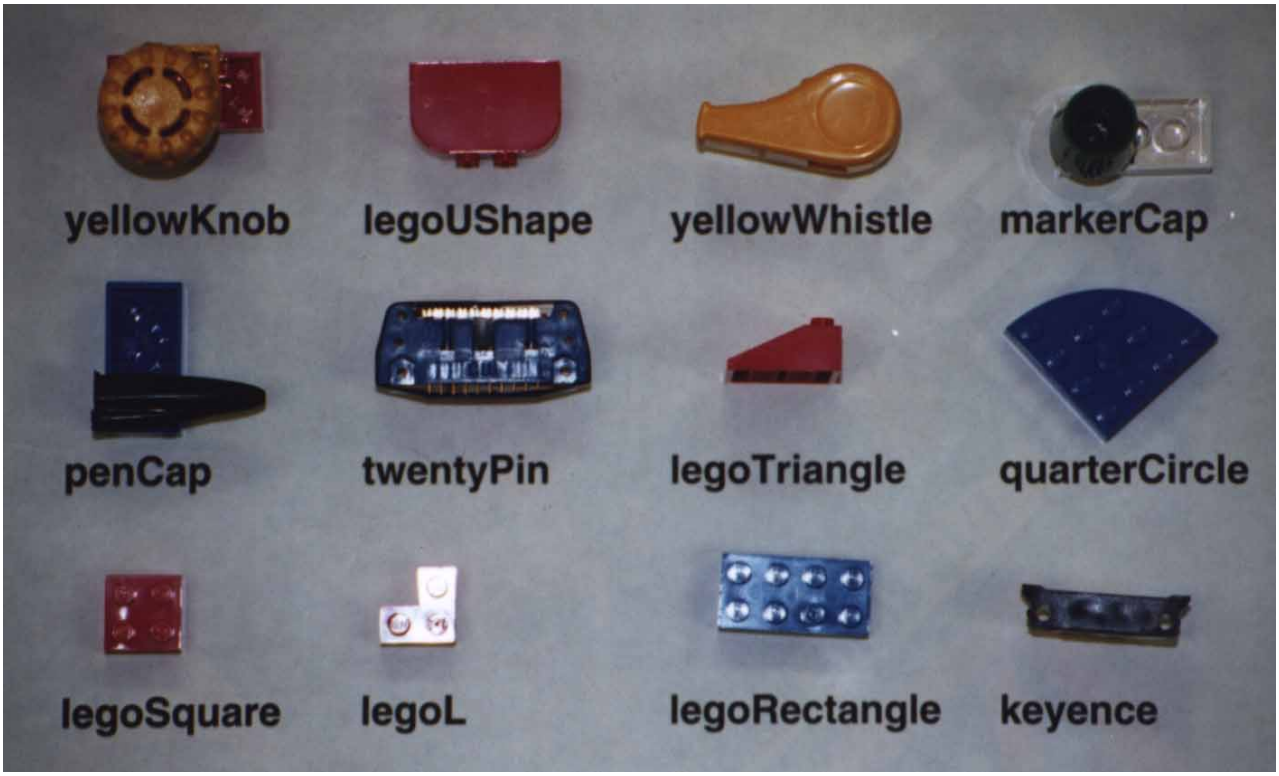


Figure 2.25: The optimal pose estimate is found by determining the transformation which optimally maps the model features (square corners) onto the sensed features (parallelepiped edges).

2.8 Experiments and Results

We experimentally analyed the technique's recognition and localization performance, and we describe those experiments in this section. The repeatability of the technique's position and orientation estimates were measured by localizing objects at known relative poses. The recognition performance was tested by identifying known objects. For these experiments, we used the online correspondence algorithm. The positional accuracy was analyzed by repeatedly localizing a lego with a $\frac{5}{8}''$ square cross-section, and the orientational accuracy was tested using a lego with a $\frac{5}{8}'' \times \frac{15}{16}''$ rectangular cross-section. The technique's recognition performance was tested using a library of twelve objects shown in Figure 2.8, where Figure 2.26 depicts the models of the objects. The objects moved relative to the crossbeam sensor at a rate of 5.0 millimeters per second in order to achieve highly accurate break point data.



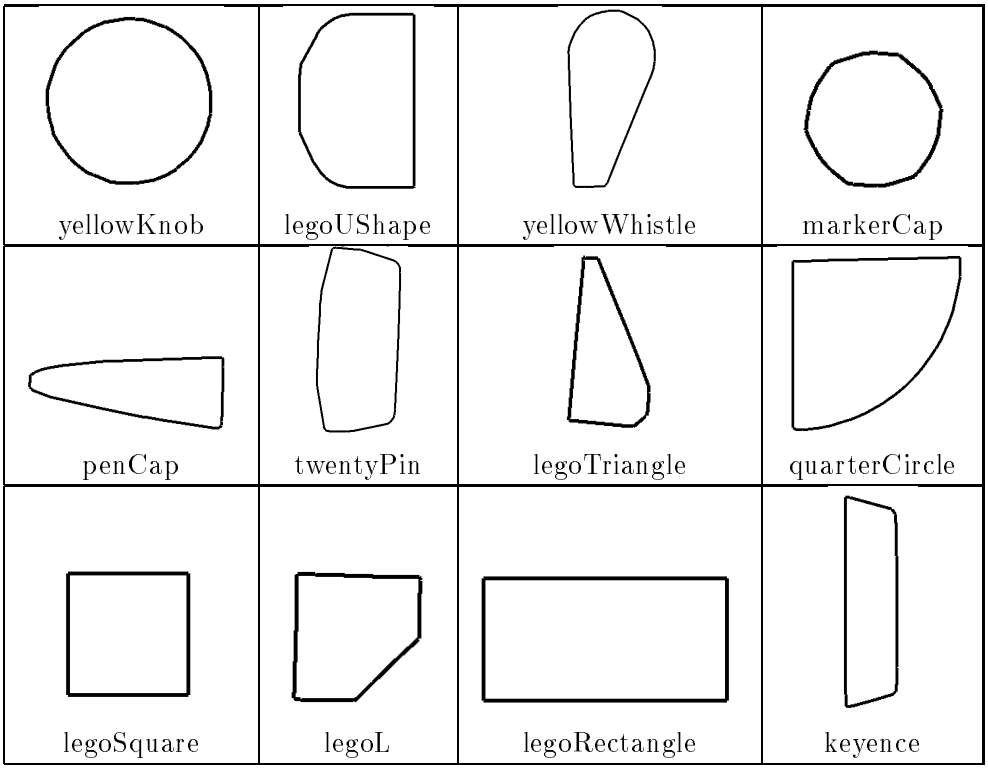


Figure 2.26: The generalized polygonal convex hull models of the objects

2.8.1 Positional Accuracy Results

The lego square with a $\frac{5}{8}''$ cross-section was passed through the crossbeam sensor along different paths in y. We assume the object's position relative to the gripper remained constant since the lego remained firmly grasped in the end effector throughout the experiment. To analyze the positional repeatable positional accuracy the object was localized multiple (25) times along each path. The standard deviations σ of the pose estimates for each path were less than 12 microns. A 2σ error bound implies that the position measurements to be repeatable to 25 microns (Table 2.4). One explanation of this accuracy assessment is that the localization technique is as highly accurate as the underlying hardware system (our Robotworld system is repeatably accurate to 25 microns).

Path X Position (mm)	Mean Localized (X,Y) Position (mm) Relative to Gripper	Standard Deviation σ of Localized Positions (mm)
98.9995	(-0.5907 , 0.2895)	0.0084
102.4989	(-0.6055 , 0.2850)	0.0087
105.9994	(-0.6032 , 0.2915)	0.0064
109.4998	(-0.6027 , 0.2867)	0.0074
112.9992	(-0.5811 , 0.2876)	0.0073
116.4997	(-0.5965 , 0.2870)	0.0097
119.9991	(-0.6043 , 0.2886)	0.0074
123.4996	(-0.6146 , 0.2936)	0.0092
126.9990	(-0.6038 , 0.2945)	0.0125

Table 2.4: The mean localized positions, and the standard deviations of those (25) localized positions for localizing a lego with a $\frac{5}{8}''$ cross-section was localized along multiple paths.

2.8.2 Orientational Accuracy Results

The localization technique's orientational accuracy was analyzed by localizing a rectangular lego with a $\frac{5}{8}'' \times \frac{15}{16}''$ rectangular cross-section at known relative orientations. The computed orientation in the end effector and the standard deviations of the estimated orientations are given in Table 2.5. The orientation estimates varied by less than 0.8°. This orientational error estimate may be due to the fact that the corners of the lego object were not perfectly crisp.

Mean commanded $\theta(\ ^o)$	Mean computed $\theta(\ ^o)$	θ offset $(\ ^o)$	Standard Deviation $\sigma\ (\ ^o)$
149.784	149.691	0.093	0.056
-30.084	-30.051	0.033	0.049
119.881	119.877	0.004	0.061
-60.023	-59.971	0.052	0.066
89.870	90.026	-0.156	0.022
-90.079	-90.166	-0.087	0.029
59.844	59.641	0.203	0.039
-120.082	-119.776	0.306	0.030
29.942	29.768	0.174	0.028
-150.069	-149.801	0.268	0.019
-0.121	0.458	-0.579	0.027
-179.730	-180.042	-0.312	0.283

Table 2.5: The commanded orientations, estimated orientations, discrepancies between the commanded and estimated orientations, and the standard deviations of those (25) orientation discrepancies for a rectangular lego with a $\frac{5}{8}'' \times \frac{15}{16}''$ rectangular cross-section.

2.8.3 Recognition Experiment

The localization technique's recognition performance was analyzed by identifying known objects using the crossbeam sensor. In this experiment, the model library contained the twelve objects shown in Figure 2.8. Each of the twelve objects was identified 50 times in various poses and the technique identified the correct model in all 600 recognition trials.

2.9 Future Work

In the future, we plan to expand the technique to handle a more expressive model class than generalized polyhedra. Extending the technique involves extending the generate and test formulations. Although at first glance, this problem appears difficult, it is greatly simplified by the fact that indexing is a general and effective technique for solving the correspondence problem. Indexing only requires constructing complete indexing tables, and, in this chapter, we developed a method for constructing complete indexing tables for systems with a single degree of freedom. Furthermore, this indexing table construction methodology is applicable to arbitrary systems with a single degree of freedom.

We also plan on investigating and developing more precise beam sensors, which would extend the applicability of crossbeam sensing to cover highly curved workpieces. We used commercially available beam sensors with beam aperture widths of a few millimeters.

Even if curved objects are scanned at the exact same height, the beam sensors do not provide breakpoint data repeatable to 25 microns because the individual sensors do not respond in the same manner. Consequently, our linear beam model broke down for highly curved workpieces.

Another interesting problem is the task of analytically designing sensors to optimally disambiguate a given set of candidate models. This problem was suggested by Prof. Ken Goldberg and Prof. John Canny. Assuming that the maximum difference between diameters and inset distances is a reasonable measure of how well the crossbeam sensor can disambiguate two objects, then we can define the task to be to determine the beam orientations which maximally disambiguate all pairs of models over all possible orientations.

2.9.1 Conclusion

In this chapter, we described a technique for identifying and localizing objects using a crossbeam sensor, which is composed of binary light beam sensors. This technique's success proves the validity of the RISC Robotics paradigm which favors simple tailored sensing devices and efficient algorithms, to achieve higher performance than generic techniques. The idea being that specialized sensors can achieve higher performance than generalized sensors, and that more accurate data is easier to process. This technique was specifically designed for generalized polyhedral objects, and it is based on fast, robust, precise, cost-effective sensors already found in industrial manufacturing. These results are applicable to existing light beam sensor systems already found in industry because current systems can be extended by incorporating computational resources to provide positional and identity information.

Chapter 3

Object Recognition and Localization from Scanning Beam Sensors

Abstract

Model based object recognition and object localization are fundamental problems in industrial automation. We present recognition and localization techniques and experimental results which use a scanner composed of binary light beam sensors to quickly recognize objects (as fast as 5 microseconds) and to accurately localize objects (0.025 millimeters). The binary scanning sensors detect only whether the part obstructs a light beam. The sensors high performance in terms of speed and accuracy is a consequence of the sensor's simple specification. Our scanning sensor is reliable, inexpensive, compact, precise, and insensitive to ambient light, which are all prerequisites for industrial manufacturing. Objects are scanned by passing them through a curtain of light beam sensors, and recording the position where the sensor outputs changed. Interpreting the resulting data involves matching the sensed data to model features, which is termed the correspondence problem; we solve the correspondence problem via a constant-time indexing technique which involves looking up the interpretation in a precomputed indexing table.

Indexing is a general approach to solving the correspondence problem in which all predicted interpretations are stored in a table indexed by the sensor data. Complete indexing tables are crucial for sparse sensing strategies such as scanning beam sensing because each

experiment produces only a single indexing vector. In this chapter, we describe a method for constructing complete indexing tables by reducing it to the task of enumerating all cells formed by an arrangement of correspondence boundaries discretization boundaries.

3.1 Introduction

Model based object recognition and model based object localization are fundamental problems in machine vision and industrial automation. Model based recognition deals with identifying a model O given a set of models $\{O_1, O_2, \ldots, O_n\}$, and model based object localization deals with estimating an object's pose from an object model and sensed data. In this chapter, we discuss efficient techniques for solving both these problems for scanning sensor data.

Scanning sensor data can be produced by moving an object relative to a scanning sensor where the sensor corresponds to an array of beam sensors arranged in a curtain formation. The positions where the sensor's output change are termed scanline endpoints (Figure 3.1). Given the scanline endpoint data and a set of modeled objects, the task is to identify the scanned object and estimate its pose (Figure 3.2). What makes this problem difficult is that scanning sensors perceive only a few boundary points from the object's silhouette. On the other hand, this problem is solvable because of the sensor's high precision (0.025 mm), and because we make some simplifying assumptions: we assume that objects are singulated and that objects are either flat generalized polygons, or generalized polyhedra with one face of the the convex hull stably resting on a horizontal surface; these assumptions reduce the system to a finite number of two-dimensional problems. In an industrial application, the environment is usually structured enough to satisfy reasonable constraints, especially if it significantly improves performance (*i.e.,* RISC Robotics, section 2.1.1). Our scanning beam sensors (stationary and mobile) are shown in Figure 3.3.

At the heart of both the recognition problem and the localization problem is the correspondence problem: the task of interpreting the sensed data as model features. In this chapter, we present an indexing solution to the correspondence problem which utilizes offline preprocessing to achieve constant time performance (as fast as 5 microseconds). This approach involves interpreting a group sensed features using indexing, and then interpreting the remaining sensed features using the pose estimate provided by the sensed feature group. After finding an interpretation and computing the optimal pose es-

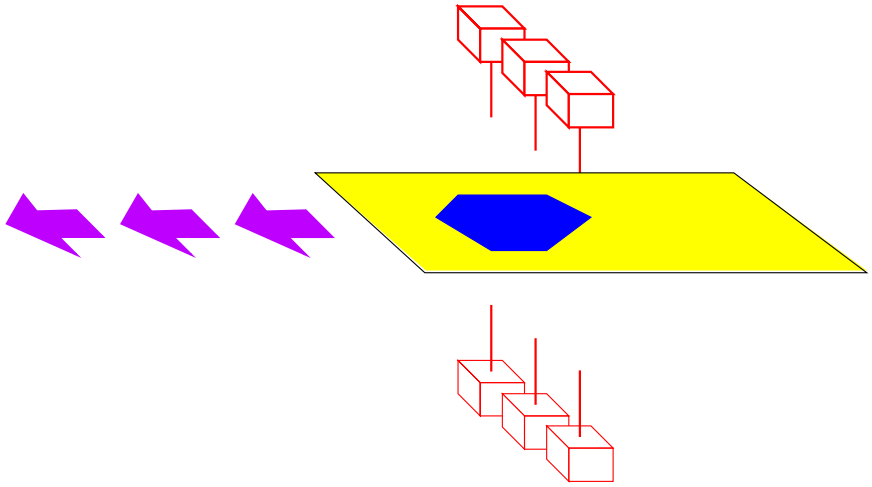


Figure 3.1: Scanning procedure

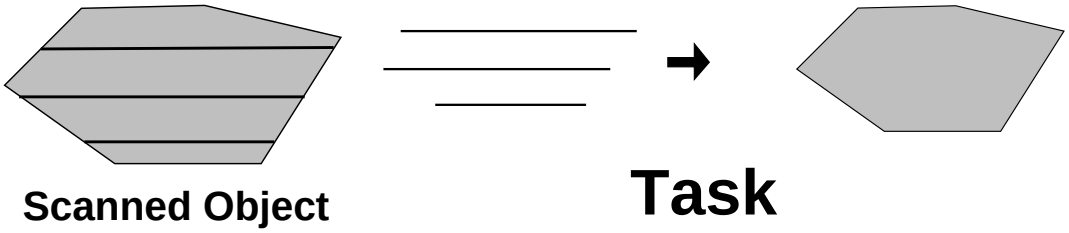


Figure 3.2: We determine the object's identity and pose from the scanline endpoints produced by scanning an arbitrary object.

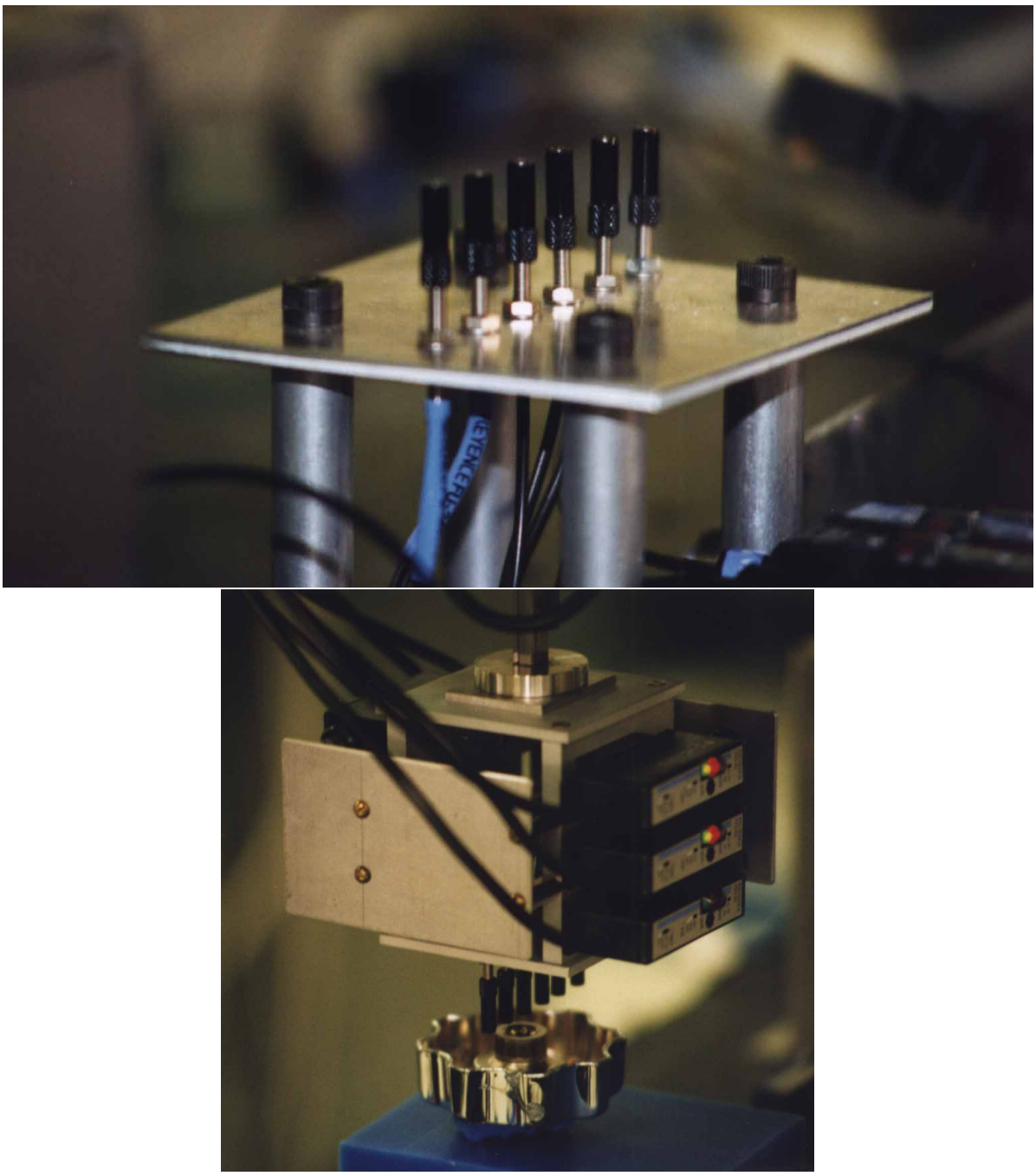


Figure 3.3: Our stationary and mobile scanning beam sensor apparati

timate in constant time, additional scanline endpoints are matched to model features in constant time. Interpretations are then revalidated via localization, by recomputing the optimal pose estimate for all of the sensed features. The scanning sensor data is so precise that pose estimates are repeatably accurate to within 0.025 mm and 0.1^o [WC95b; WM94b]. For comparison, commercial machine vision systems recognize and localize objects via template matching to sub-pixel accuracies in sixty milliseconds.

Performance is the main reason for using scanning beam sensors; other considerations include robustness, size, speed, ease of calibration, and monetary cost. Commercially available binary beam sensors are repeatably accurate to less than 0.025 mm, and, in addition, are robust, inexpensive, and insensitive to ambient lighting. Furthermore, our correspondence and localization approaches are applicable to other sensors and techniques which follow the scanning paradigm: interpreting scanline camera data, interpreting continuous output from three dimensional range sensors, and registering objects from contact probe data.

3.1.1 Indexing Overview

In this chapter, we describe an indexing-based approach for solving the correspondence problem from scanline data, which was originally described in [WC95b]. The term indexing (section 2.1.4) describes a generic approach to model-based recognition which relies upon offline preprocessing. Indexing techniques involve extracting indexing coordinates from tuples of sensed features, and quantizing these coordinates to achieve integral indices to index table entries which contain the consistent interpretations [CJ91; LSW88; KSSS86; FMZ+91]. The main advantage of indexing techniques is that a single lookup compares a sensed feature tuple with all of the model feature tuples' this is done by merging the indexing tables for all model feature tuples into a composite table. Figure 3.4 depicts the entire indexing process: the data is extracted and discretized, and then the discretized data is used to index into a table entry containing all valid feature correspondences.

An indexing table provides a map from predicted indexing coordinates to predicted interpretations. Indexing relies upon the assumption that the predicted measurements and the actual measurements are discretized to the same quantized coordinates; consequently, the indexing table should provide a map from experimentally measured indexing coordinates to consistent interpretations. Furthermore, indexing relies upon the fact that only a small

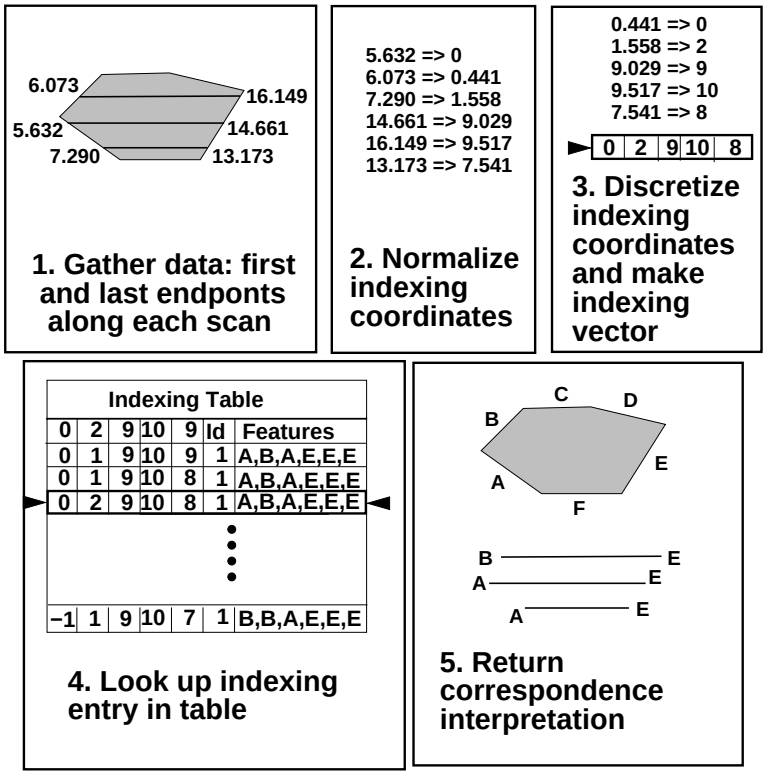


Figure 3.4: Indexing techniques involve discretizing the sensed data to index the entry containing valid interpretations.

fraction of all indexing coordinates is consistent with each model feature group; otherwise, a large fraction of model feature groups may match a sensed feature group, which will reduce the performance of the technique to a brute force generate and test approach.

Constructing Complete Indexing Tables

Completeness is a critical issue for sparse sensing applications, such as scanning beam sensing, because each experiment may only provide a single indexing vector. Previous indexing methods did not guarantee complete tables, except in degenerate cases; for sparse sensing strategies, a new approach to constructing indexing tables is required. Incomplete indexing tables were satisfactory for dense sensor strategies, incomplete tables are unsuitable for sparse sensing strategies because each experiment may produce only a single indexing vector, and an incomplete indexing table would have drastic consequences. Incomplete indexing tables would result in unidentified or misinterpreted objects, and this shortcoming would probably make the technique unacceptable for industrial applications.

We implemented the complete indexing table construction method originally described by Wallack and Canny [WC95a]. Complete indexing tables also enable us to estimate the probability that objects cannot be uniquely identified by checking for intersection between indexing tables.

In this chapter, we describe an approach for constructing complete indexing tables for sparse sensing strategies. This approach enumerates indexing table entries by enumerating cells in an arrangement in configuration space [Ede87]. The key idea is that the process of quantizing the indexing coordinates to achieve integral indices corresponds to partitioning indexing space into equivalence classes of discretization hyperplanes. Therefore, a complete indexing table corresponds to a regular cell covering of the predicted manifold. This observation serves as a basis for an efficient algorithm for enumerating all of the indexing table entries. Each indexing table entry corresponds to a finite number of continuous regions (cells) in configuration space, and the cells are defined by only two classes of boundaries: discretization boundaries separating configurations which are discretized to different indexing coordinates, and correspondence boundaries separating configurations with different correspondence interpretations.

3.1.2 Outline

The rest of the chapter is organized in the following manner. In Section 2 we describe the complete indexing table construction methodology. In Section 3, we describe the theoretical framework for the scanning sensing technique. In Section 4 we detail the complete construction of the indexing table. In section 5, we extend this construction technique to handle generalized polygons. The pose estimation technique is discussed in Section 6. In Section 7 we measure the performance of the scanning sensing technique, and conclude by summarizing the results and advantages of this technique.

3.2 Constructing Complete Indexing Tables

In this section, we detail the complete indexing table construction methodology.

3.2.1 Correspondence Problem

To clearly describe the complete construction methodology, we first describe the correspondence problem geometrically. Furthermore, we focus on a simple sensor system: a bounding box sensor, in order to precisely define the correspondence problem. The bounding box sensor perceives the height and width of the rectangle containing a two-dimensional object in an arbitrary pose (Figure 3.6), and two object models (Figure 3.5). In this case, the correspondence problem involves determining the model features (vertices) which correspond to the four sides of the bounding box, or, in other words, to map from sensor data to consistent interpretation(s) (Figure 3.7). In Figure 3.8, one valid interpretation (consistent with the given height and width) is *(B,B,C,C)*, which describes the situation where right rectangle edge contacts vertex B, the top rectangle edge contacts vertex B, the left rectangle edge contacts vertex C and the bottom rectangle edge contacts vertex C.

3.2.2 Indexing Example

The sensor perceives width and height, providing a two-dimensional indexing space. Since the bounding-box is invariant with respect to translation, the only remaining degree of freedom is the orientation θ. For each model, each orientation θ corresponds to a single point in indexing space, and the union of all predicted indexing values is termed

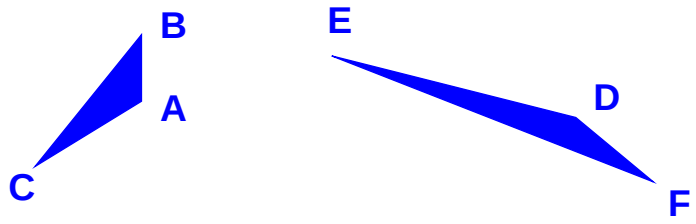


Figure 3.5: Two object models for the correspondence problem

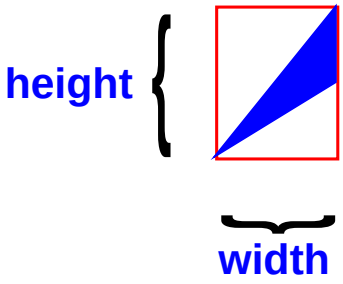


Figure 3.6: The rectangle sensor perceives the rectangle containing an object at a given orientation

the *model stratification*. We can imagine coloring each point on the model stratification according to the correspondence interpretation (Figure 3.9).

From a geometric perspective, solving the correspondence problem amounts to determining the model stratification closest to the given data (Figure 3.10). The closest stratification could be found by characterizing each stratification and computing the distance between the data point and each stratification, but this approach has the drawback of having to compute minimum distances between points and stratifications.

A simpler method is necessary to achieve constant time performance. Indexing usually involves quantizing the sensor values, and using the integral coordinates to index an entry in an indexing table containing the consistent interpretations. Discretizing the data in this way partitions indexing space into equivalence classes of discretization hypercubes, where every point in a discretization hypercube is quantized to the same integral coordinates. Since each model stratification is of finite size, they only cover a finite number of discretization hypercubes, and since there are only a finite number of interpretations, each model feature group corresponds to a finite number of indexing table entries. In the next section, we explain how we enumerate all of these different indexing table entries.

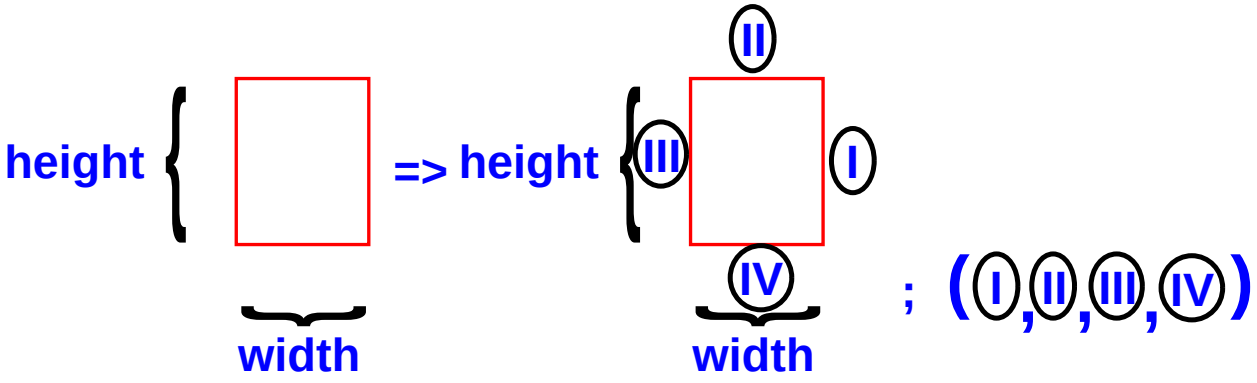


Figure 3.7: The correspondence problem entails determining the model features corresponding to the sensed data

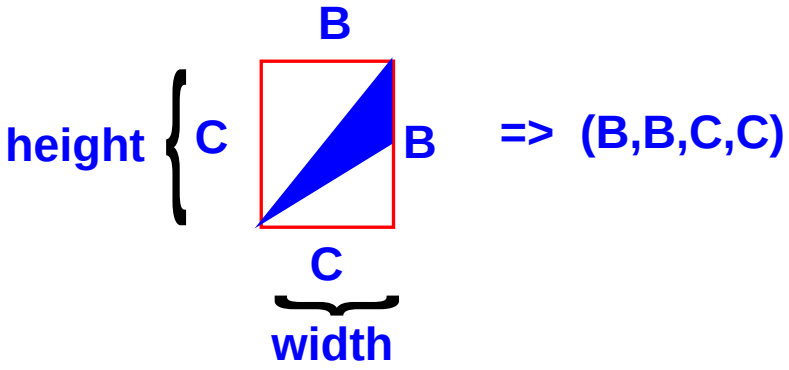


Figure 3.8: An example correspondence, the height and width may correspond to the features B, B, C, C

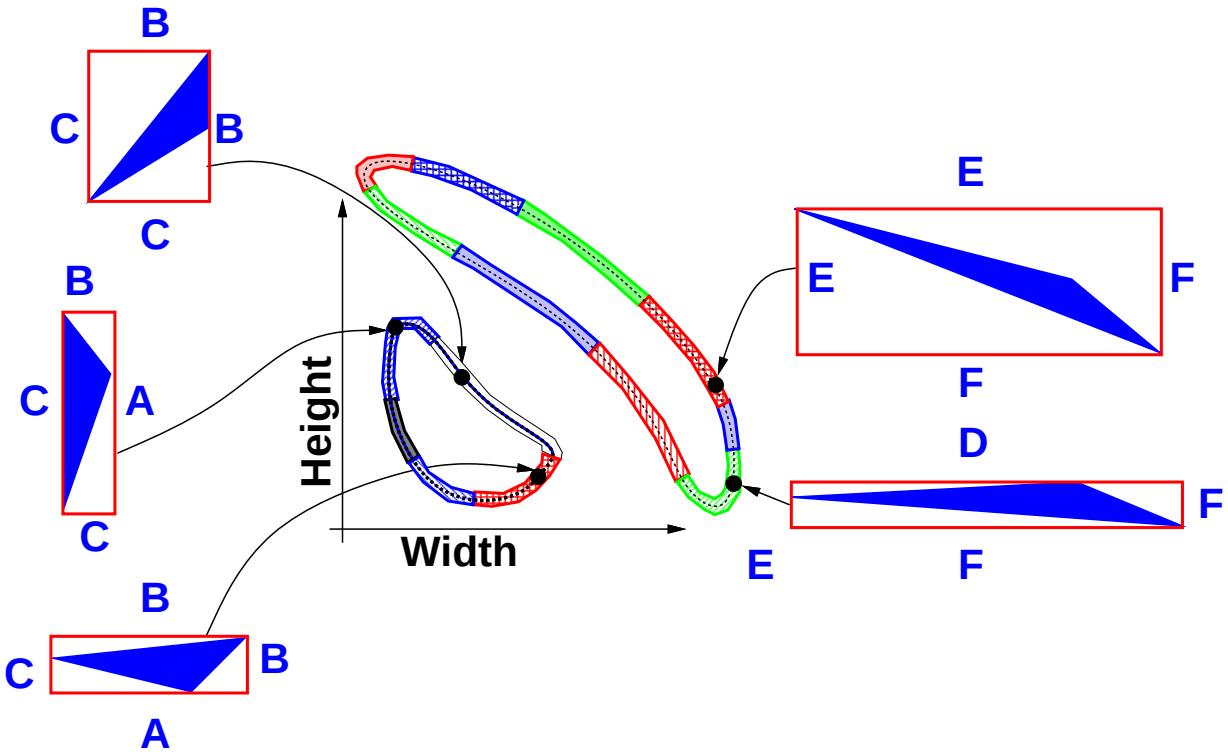


Figure 3.9: Two one-dimensional stratifications for two objects in a two-dimensional indexing space for a rectangle sensor

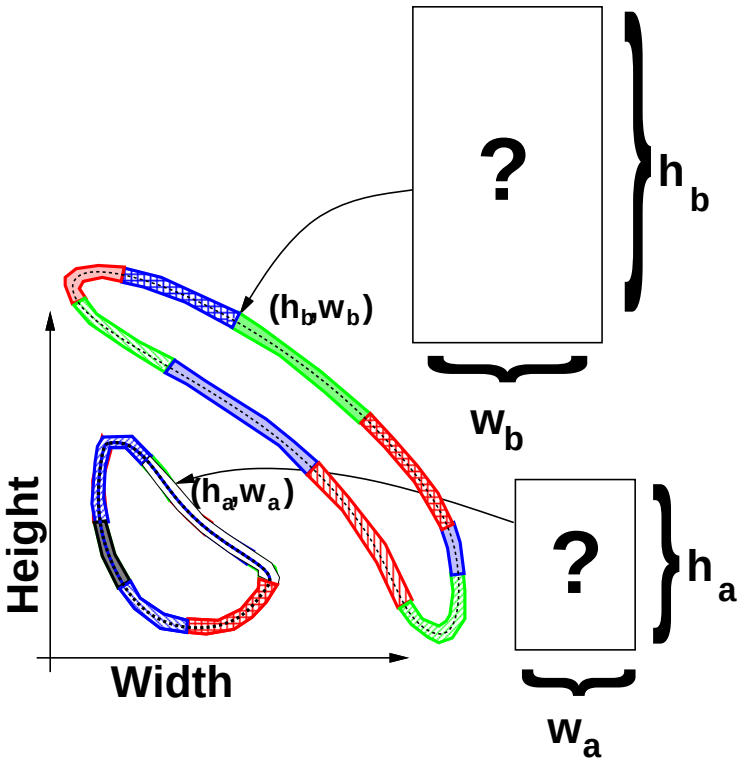


Figure 3.10: Object identification is achieved by finding the predicted data which best matches the experimentally sensed data.

3.2.3 Overview of Construction Strategy

The technique for constructing complete indexing tables is based upon two reductions: first, the task of constructing a complete indexing table is reduced to the problem of enumerating a complete regular cell covering of a model stratification in where each cell corresponds to a discretization hypercube and a consistent interpretation, and, second, the problem of enumerating a complete cell covering of the stratification is reduced to the problem of completely enumerating all of the cells in an arrangement in configuration space of discretization boundary curves and correspondence boundary curves. These two reductions yield an efficient method for constructing complete indexing tables.

We now explain the rationale behind these reductions. Recall that quantizing the indexing coordinates is analogous to partitioning indexing space into equivalence classes of discretization hypercubes. Each distinct indexing table entry corresponds to a unique combination of discretized indexing coordinates and a correspondence interpretation. Constructing a complete indexing table amounts to enumerating indexing table entries for all different discretized indexing coordinates and all of the consistent interpretations.

We now prove the correctness of this technique. Consider the continuous patches on the model stratification where the discretized indexing coordinates remain constant, and the interpretation remains constant. Notice that the entire patch corresponds to a single indexing entry. We can also view this patch in configuration space; *i.e., each indexing table entry corresponds to at least one continuous patch of configuration space.* Since the indexing entries only depend upon discretized coordinates and interpretations, the only ways that the indexing entry can change is when we cross over a boundary corresponding to different discretized coordinates or crosses over correspondence boundaries. Therefore, the configuration space patches corresponding to separate indexing table entries are equivalently the cells in the arrangement formed by the discretization boundaries and correspondence boundaries.

Constructing a complete indexing table involves enumerating at least one witness for each indexing table entry. This can be achieved by intersecting the discretization and correspondence boundary curves. We simply predict an indexing table entry at each such intersection. Alternatively, we can enumerate the cells in the arrangement via a sweep algorithm, which would be more efficient, although more complex.

3.2.4 Example: Constructing A Complete Indexing Table for a One Dimensional Stratification

Figure 3.11 shows a complete regular cell covering of the stratification for triangle ABC. Each indexing table entry is attributable either to a different discretization or a different interpretation. We can enumerate all of the different indexing table entries by enumerating at least one witness for each cell; such witnesses can be found by computing the intersections of all the model stratification with discretization boundaries and enumerating the boundary points where the correspondence interpretation changes (Figure 3.12).

Alternatively, we can enumerate the cells in configuration space by projecting the discretization boundaries and the correspondence boundaries from indexing space down onto configuration space. The indexing entries correspond to cells in the arrangement formed by these two types of curves.

This complete indexing construction technique extends to higher dimensions as well, and we will describe two dimensional generalization in more detail in section 4. The downside if this indexing technique is that the sizes of the indexing tables and the time spent constructing the se tables depends exponentially upon the dimensionality of configuration space. For this reason, it is advantageous to realize a dense characterization of the system in as few degrees of freedom as possible.

3.3 Theoretical Framework

In this section, we discuss geometric foundation, the normalization procedure, and pose parameterization.

3.3.1 Geometric Foundation

Scanning sensors perceive only a few points from a silhouette of an object (the silhouette corresponds to a projection along the axis of the scanning beams). Since the object is assumed to be either a flat generalized polygon or a generalized polyhedron stably resting on a horizontal surface, we only need to consider a finite number of two-dimensional generalized polygonal silhouettes.

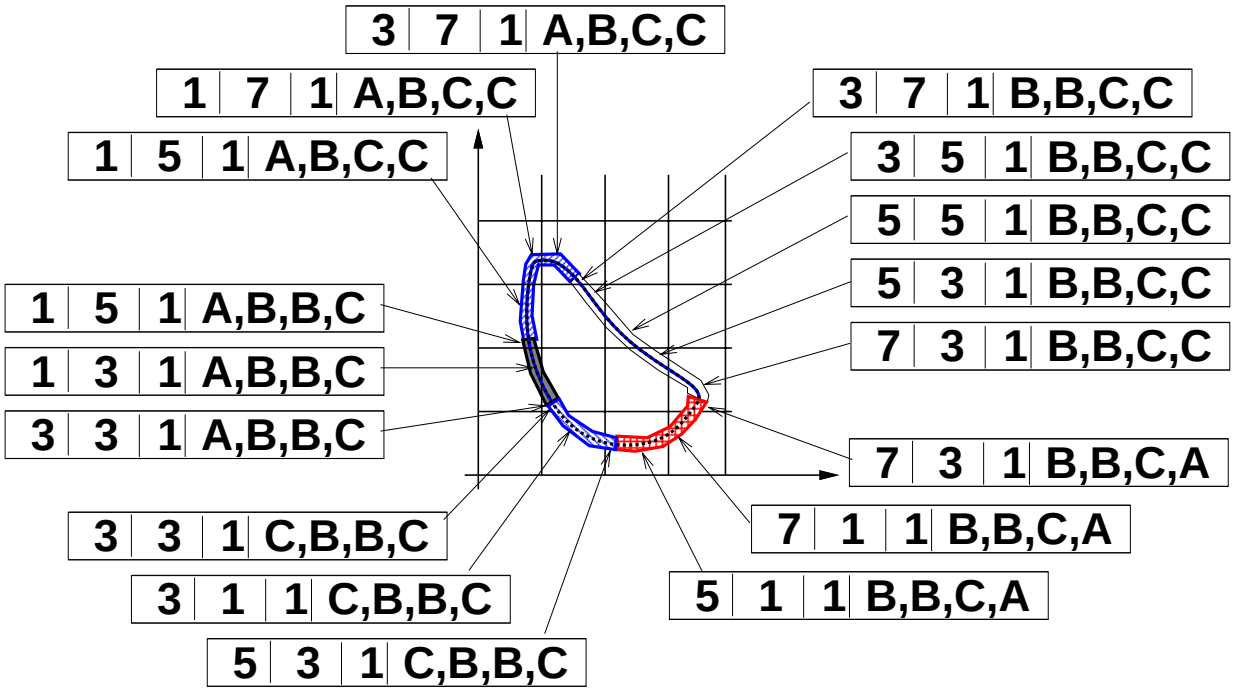


Figure 3.11: A complete cell covering for the model stratification of triangle ABC.

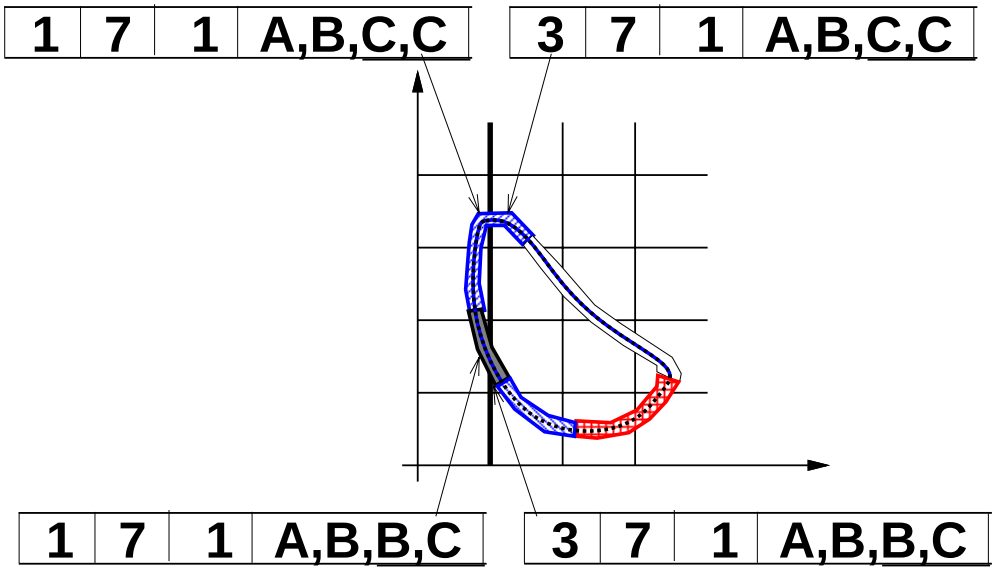


Figure 3.12: Witnesses to all of the cells can be found by intersecting the model stratification with all the discretization boundaries and enumerating all the boundaries between different correspondences.

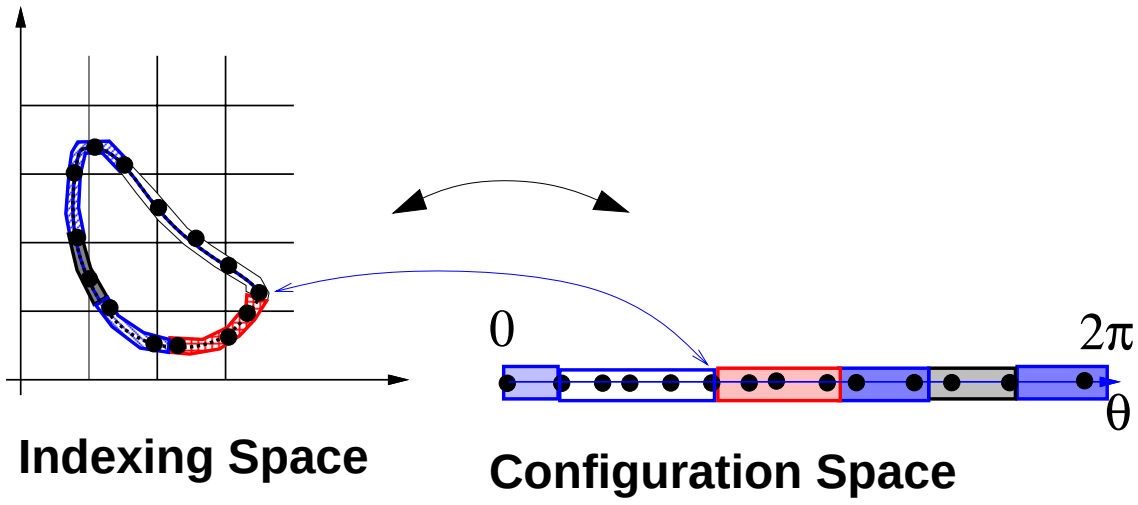


Figure 3.13: Another method of enumerating all of the cells is to consider the cell boundaries as curves in configuration space, and to enumerate all of the cells in the arrangement of those curves

3.3.2 Normalization

Normalization is a technique for reducing the effective number of degrees of freedom in order to condense the indexing table; normalization involves distilling the correspondence-dependent information and disregarding redundant or correspondence-independent information from the sensor data. Since the object's absolute position along the scanning path is irrelevant for the sake of determining the correspondences, only the relative positions of the scanline endpoints provide correspondence-dependent information. Scanline endpoints are normalized by subtracting out the position of a particular scanline endpoint, termed the reference scanline endpoint (Figure 3.14). The term *normalized scanline endpoint* ($\overline{x_i}$) refers to the position of a scanline endpoint relative to the reference scanline endpoint. p scanning beam sensors generate $2p \Leftrightarrow 1$ normalized scanline endpoints. Since the median sensor will register the object when the object is registered by at least half of the sensors, the initial scanline endpoint of the median sensor is a reasonable choice for the reference scanline endpoint.

3.3.3 Configuration

Without loss of generality, we assume the scanning sensor travels in the x direction, making the x degree of freedom correspondence independent. Therefore, the object's config-

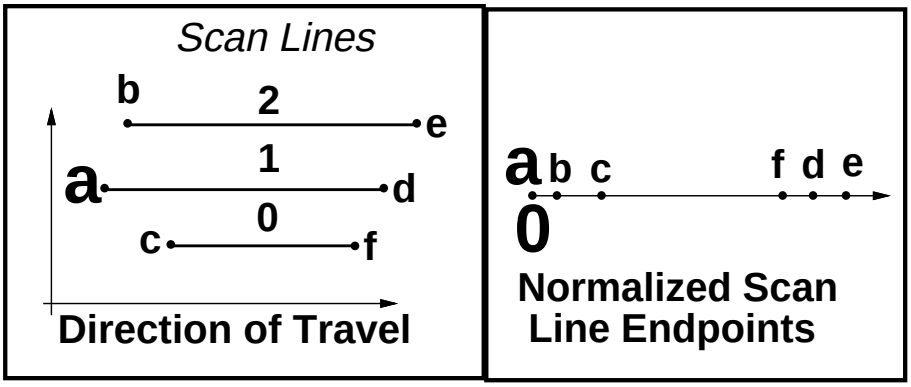


Figure 3.14: The scanline endpoints are *normalized* by subtracting the position of the reference scanline endpoint (in this case, scanline endpoint "a").

uration depends only upon y and θ. Configuration (y, θ) corresponds to rotating the object counterclockwise by θ and then translating the object in the y direction by y (Figure 3.15). Indexing entries are synthesized by predicting the discretized normalized intersections of the transformed object with horizontal scanlines (Figure 3.16). To accommodate slightly incorrect models, we include additional indexing table entries corresponding to ϵ tolerances.

3.3.4 Indexing Using Extremal Scanline Endpoints

To achieve a fixed number of indexing coordinates, only the first and last scanline endpoints from each scanning sensor are used for indexing. Internal scanline endpoints, due to either non-convex objects, or internal holes, are used to improve the localization estimate. In order to utilize additional scanline endpoints, we first need to match them to corresponding features; this can be done efficiently given the pose estimate (y, θ)

In order to achieve smaller indexing tables, we may construct the table in terms of a subset of the scanning beams. In these cases, the scan data from the primary beams provides a fine enough pose estimate to determine the model features corresponding to the other scanline endpoints. We can then use all of the scanline endpoints to realize the optimal pose estimate.

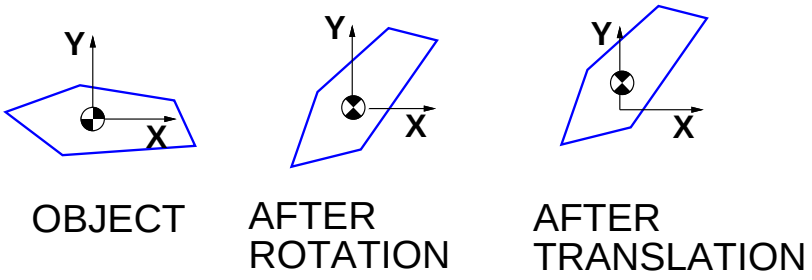


Figure 3.15: Configuration (θ, y) of object O corresponds to a copy of O rotated about the origin by θ and then translated along the y-axis by y.

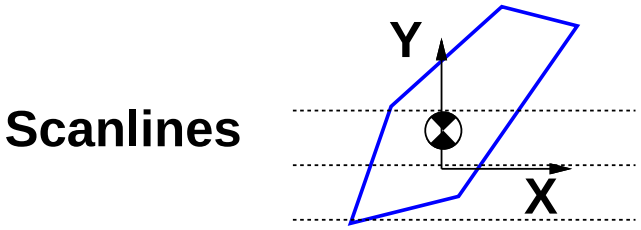


Figure 3.16: Scanning sensor data is synthesized by predicting the discretized normalized intersections of the transformed object with scanlines.

3.3.5 Determining Incident Features for a Given Pose (y, θ)

The task of synthesizing indexing entries is dual to the task of interpreting additional features (after the pose has been attained) because both tasks involve efficiently determining which feature is incident on a particular scanline.

Determining the incident feature (given (x, y, θ)) naively by intersecting the model with the scanlines takes $O(n+m)$ time; this running time can be improved to $O(\log(n)+m)$ time via geometric data structures, and it can also be computed to $O(m)$ time using additional precomputed lookup tables. The geometric approach exploits the coherence in the the orderings of the left and right projections (along the y axis) of the edges; the orderings remain constant over continuous ranges of orientations. In fact, there are only n^2 different orderings because the ordering can only change when two edge vertices attain the same y coordinate. The lookup table approach involves discretizing (θ, y) configuration space, and enumerating all of the scanline interpretations for each range of orientations $(\langle\theta_{min}, \theta_{max}\rangle, \langle y_{min}, y_{max}\rangle)$. A finely discretized table would produce $O(m)$ expected running time.

3.4 Complete Table Construction

In this section, we detail construction of complete indexing tables for scanning sensor data. The key idea is that each indexing table entry corresponds to a finite number of continuous regions in configuration space; furthermore, these regions are separated by only two types of boundaries: discretization boundaries separating configurations which are discretized to different indexing coordinates (Figures 3.17,3.20), and correspondence boundaries separating configurations with different correspondences (Figures 3.18,3.24). Boundaries are curves in this (y, θ) two-dimensional configuration space. We can enumerate all of the indexing table entries by enumerating all of these regions in configuration space; one way to enumerate all of the regions is to enumerate all of the pairwise intersections of the boundary curves.

3.4.1 Algorithm Outline

Definition 3.1 *Discretization* refers to the process of rounding an indexing coordinate to the closest multiple of the discretization $\delta : \overline{x_i} \rightarrow \lfloor \frac{\overline{x_i}+\frac{\delta}{2}}{\delta} \rfloor$

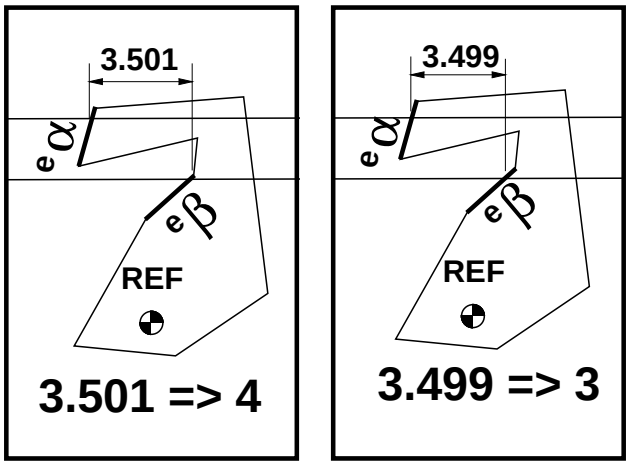


Figure 3.17: Two different configurations with different indexing coordinates; the normalized scanline endpoints are discretized to different indexing coordinates.

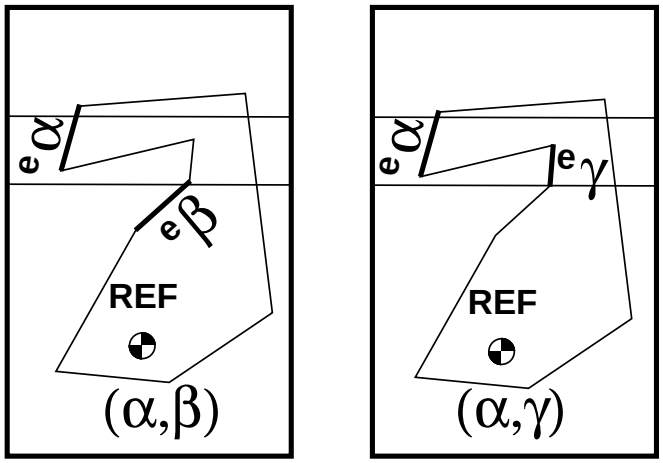


Figure 3.18: Two different configurations with different feature correspondences.

Definition 3.2 *Discretization Boundary Curves* characterize configurations for which a predicted normalized scanline endpoint straddles two discretization regions: $\overline{x_i} = k\delta + \frac{\delta}{2} | k \in \mathbb{N}$.

Definition 3.3 *Correspondence Boundary Curves* characterize configurations for which a predicted normalized scanline endpoint straddles two correspondence interpretation regions; *i.e.*,, the scanline contacts two features at the incident vertex.

Complete indexing tables are constructed in three steps:

1. Enumerate all discretization boundary curves and correspondence boundary curves
2. Enumerate all intersections between these curves
3. Synthesize indexing table entries at intersections

Figure 3.19 depicts a set of discretization boundary curves and correspondence boundary curves in configuration space. Each region (cell in the arrangement) corresponds to a single indexing table entry. A complete indexing table is produced by synthesizing an indexing table entry for each configuration where two curves intersect; in order to compute intersections algebraically, we use a change of variables from θ to t ($t = \tan(\frac{\theta}{2})$) to transform trigonometric expressions into rational algebraic expressions.

Discretization Boundary Curves

Discretization boundary curves ($DBC(\theta, y)$, DB-curves) describe configurations in which one of the normalized scanline endpoints is exactly $\overline{x_\alpha} = k\delta + \frac{\delta}{2}$, where $k \in \mathbb{N}$, or $x_\alpha \Leftrightarrow x_{Ref} = k\delta + \frac{\delta}{2}$. For example, Figure 3.20 depicts a DB-curve corresponding to edges e_α, e_β with discretization $x_\alpha \Leftrightarrow x_{Ref} = 3.5$. DB-curves are defined in three steps.

DB-curves are formulated via two intermediate functions. First we formulate the x coordinate of the extended intersection between a horizontal scanline and an arbitrary edge $\mathcal{E}_\alpha$ as a function $I(\theta, y)$ of θ and y (Figure 3.21 and equation (3.1)). Second, we formulate the difference between two extended intersections $\Delta I(\theta, y)$ as a function of θ and y; and third, we define DB-curves as the set of all configurations for which the difference function is equal to the desired value $DB_{\beta,\alpha}(\theta, y)$ in terms of θ and y.

In Figure 3.21, $\mathcal{E}_\alpha$ is parameterized by α and R_α, where α refers to $\mathcal{E}_\alpha$'s outward pointing normal direction, and R_α refers to the minimum distance from the origin to $\mathcal{E}_\alpha$. The horizontal scanline is characterized by its height D_α.

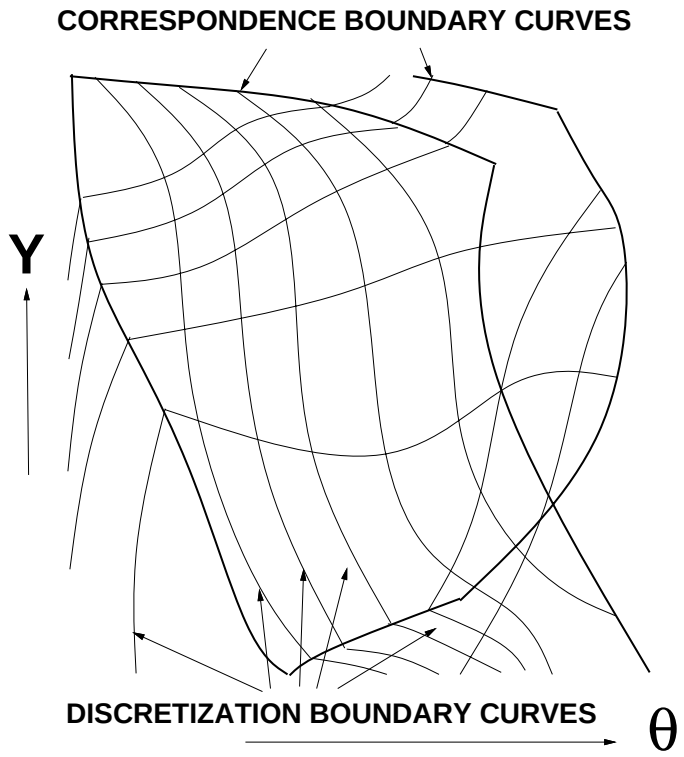


Figure 3.19: A set of discretization boundary curves and correspondence boundary curves in (θ, y) configuration space.

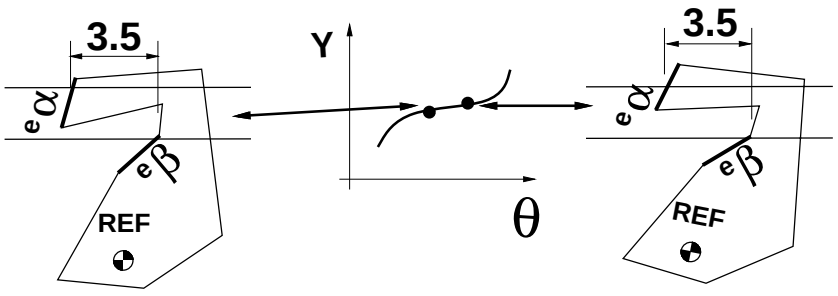


Figure 3.20: A DB-curve corresponding to edges e_α, e_β with discretization $x_\alpha \Leftrightarrow x_{Ref} = 3.5$.

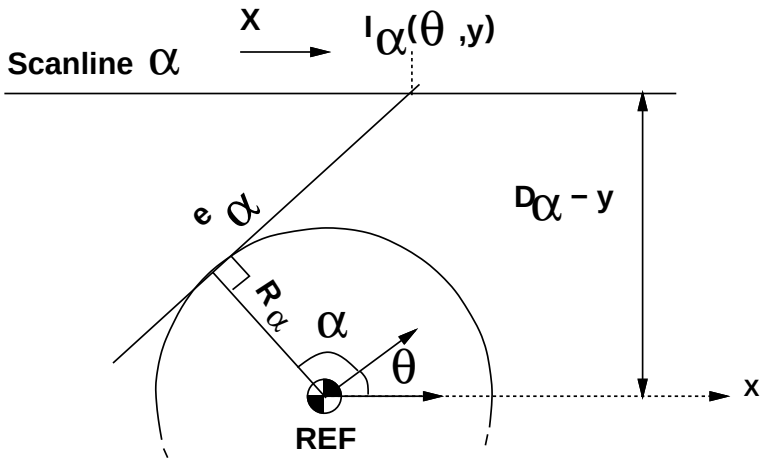


Figure 3.21: Extended intersection function $I(\theta, y)$ characterizes the x coordinate of the extended intersection between transformed edge segment $\mathcal{E}_\alpha$ and scanline α.

$$I_\alpha(\theta, y) = \frac{R_\alpha \Leftrightarrow (D_\alpha \Leftrightarrow y)\sin(\theta + \alpha)}{\cos(\theta + \alpha)} \tag{3.1}$$

Extended intersection difference functions (ΔI) characterize the difference between extended intersection functions (Figure 3.22 and equation (3.2)). Discretization boundary curves are the set of configurations where the extended intersection difference function is equal to the desired value, or, in other words, the zero set of the difference between the extended intersection difference function and the desired value, $\Delta_{\beta,\alpha} = \Leftrightarrow k\delta + \frac{\delta}{2}$. Cross multiplying by the denominators of the two extended intersection difference functions yields an algebraic expression, $DBC(\theta, y)$ (equation (3.3)).

$$\Delta I_{\beta,\alpha}(\theta, y) = I_\beta(\theta, y) \Leftrightarrow I_\alpha(\theta, y) \tag{3.2}$$

$$DBC_{\beta,\alpha}(\theta, y) = (\Delta I_{\beta,\alpha}(\theta, y) \Leftrightarrow \Delta_{\beta,\alpha}) \tag{3.3}$$

3.4.2 Orientation Ranges of DB-Curves

Although the DBC equations correspond to infinitely long lines, DB-curves correspond to finite length edge segments. The endpoints of the edge segments restrict the orientation range of DB-curves (Figure 3.23), and this orientation range depends solely on the edges and the offset between the discretized contact point and the origin ($\Delta_{\beta,\alpha}, D_\beta \Leftrightarrow D_\alpha$).

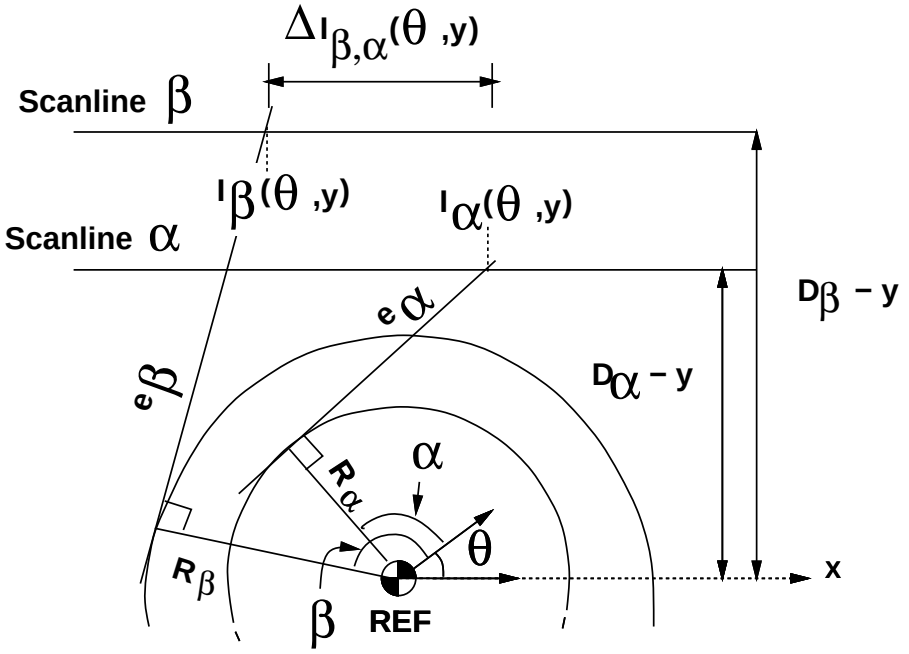


Figure 3.22: Extended intersection difference functions $\Delta I_{\beta,\alpha}(\theta, y)$.

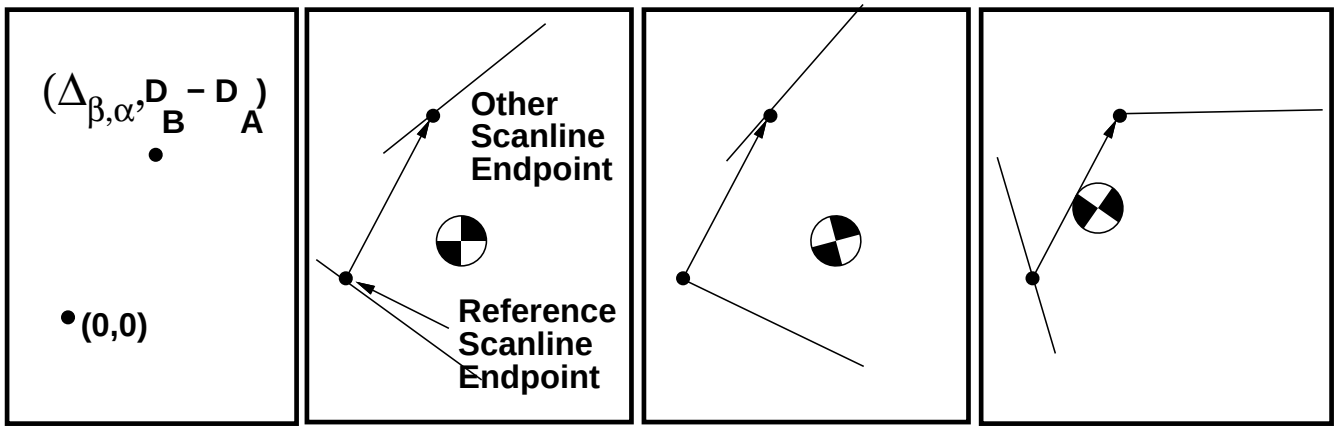


Figure 3.23: The actual endpoints of the edge segments restrict the orientation range of DB-curves.

Correspondence Boundary Curves

Correspondence boundary curves ($CBC(\theta, y)$, CB-curves) characterize configurations for which the scanline intersects two features at the incident vertex. For example, Figure 3.24 depicts a CB-curve corresponding to a scanline intersecting vertex v_γ; the CB-Curves depend only upon the vertex v_γ, and the scanline height S (Figure 3.25 and equation (3.4)). Every CB-curve spans the entire orientation range $[0, 2\pi\rangle$ because a rotated vertex can always be translated onto the scanline via translation in y.

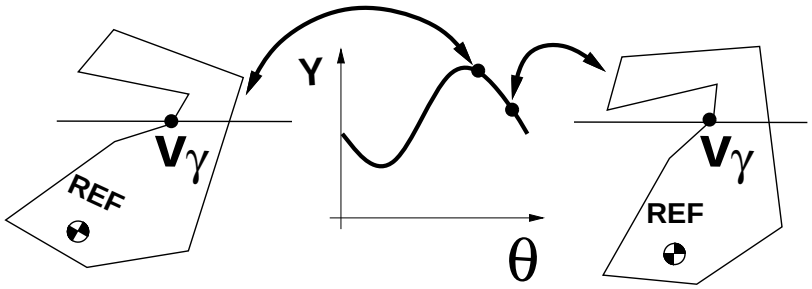


Figure 3.24: Correspondence Boundary Curves (CB-curves) characterize the configurations for which the scanline contacts two model features at the incident vertex v_γ.

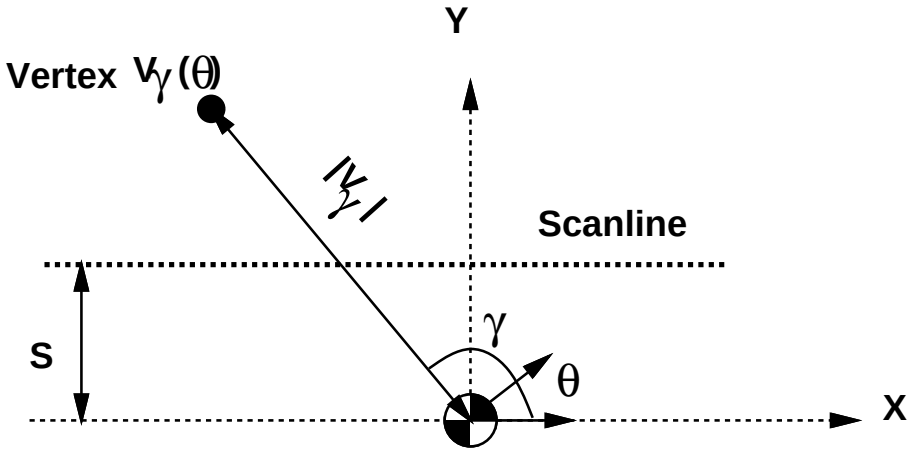


Figure 3.25: CB-curves only depend upon the vertex position V_γ and the scanline height S.

$$y(\theta) = |V_\gamma| \sin(\theta + \gamma) \Leftrightarrow S \tag{3.4}$$
$$CBC_\gamma(\theta, y) = |V_\gamma| \sin(\theta + \gamma) \Leftrightarrow S \Leftrightarrow y$$

3.4.3 Computing Curve Intersections

In this section, we describe algebraic and numerical methods for computing the intersections of DB-curves and CB-curves.

Intersecting DB-Curves

This section describes how we compute the intersections of DB-curves (without loss of generality, α, β, γ are the edges and α is the reference edge $DBC_{\beta,\alpha}(t,y) = DBC_{\gamma,\alpha}(t,y) = 0$). In the general case, all three of the edges are distinct. The degenerate case corresponds to two of the edges being parallel; in this case, the orientation θ can be computed by finding the orientation for which the rotated points attains the desired difference in y.

The general case is handled by intersecting the two curves algebraically. We exploit the fact that two of three y dependent terms in both $DBC_{\beta,\alpha}(t,y)$ and $DBC_{\gamma,\alpha}(t,y)$ cancel each other out leaving $y\sin(\alpha \Leftrightarrow \beta)(1+t^2)^2$ and $y\sin(\alpha \Leftrightarrow \gamma)(1+t^2)^2$ as the only y-dependent terms. Cross-multiplying $DBC_{\beta,\alpha}(t,y)$ by $\sin(\alpha \Leftrightarrow \gamma)$ and $DBC_{\gamma,\alpha}(t,y)$ by $\sin(\alpha \Leftrightarrow \beta)$ results in both functions having the same y-dependent term which can be subtracted from both sides of the equation; therefore the difference between these two expressions is an expression solely in t (equation (3.5)). Further simplification results in a quadratic expression in t which can be solved exactly. This method of removing variables time from a multivariate system is termed *elimination*; although this method is tailored to this particular problem, there are general techniques which can eliminate arbitrary variables from arbitrary multivariate systems, with the intention of reducing the system to a univariate system which can be solved exactly. After determining each plausible orientation t, we can compute the y translation and verify each candidate solution.

$$
\begin{aligned}
\underbrace{\sin(\alpha-\beta)y(1+t^2)^2 + DBC^{*y}_{\beta,\alpha}(t)}_{DBC_{\beta,\alpha}(t,y)} &= 0 = \underbrace{\sin(\alpha-\gamma)y(1+t^2)^2 + DBC^{*y}_{\gamma,\alpha}(t)}_{DBC_{\gamma,\alpha}(t,y)} \\
\sin(\alpha-\gamma)[\sin(\alpha-\beta)y(1+t^2)^2 + DBC^{*y}_{\beta,\alpha}(t)] &= 0 = \sin(\alpha-\beta)[\sin(\alpha-\gamma)y(1+t^2)^2 + DBC^{*y}_{\gamma,\alpha}(t)] \\
\Rightarrow -\sin(\alpha-\beta)DBC^{*y}_{\beta,\alpha}(t) &= -\sin(\alpha-\gamma)DBC^{*y}_{\gamma,\alpha}(t) \\
\Rightarrow a_2t^2 + a_1t + a_0 &= 0 \qquad (3.5)
\end{aligned}
$$

Intersecting CB-Curves and DB-Curves

The orientation of the intersection of DB-curves and CB-curves is found by substituting equation (3.4) into equation (3.3), and cross multiplying producing an equation solely in t which can be solved numerically. The y translation is computed given t.

Intersecting CB-Curves

The orientations θ of the intersection of two CB-curves are determined by finding the two orientations for which the vector between the vertices has the desired y component. The y translation is computed given θ.

3.4.4 Complete Lookup Table Sizes

Complete indexing tables were constructed for multiple objects at various discretizations. For various polygonal objects shown in Figure 3.26, Table 3.1 lists the sizes of indexing tables, and κ, the average number of interpretations for each valid indexing coordinate, $\sigma(\kappa)$, the standard deviation of the number of interpretations, and the table construction time. We constructed the indexing table using three scanning beams ($m = 3$). The running times could have been decreased had we implemented either the $O(m \log(n))$ method or $O(m)$ method for synthesizing predicted indexing coordinates. Notice that the number of indexing table entries increases as one over the square of the discretization because we are dealing with a two degree of freedom system. We verified the completeness of the indexing tables by constructing finely discretized tables (2.5 mm) and generating data for 100 simulated scanning and validating that the table included the corresponding table entry.

3.5 Complete Table Construction for Generalized Polygons

The scanning sensor technique is capable of handling generalized polygons. The original localization algorithm was developed to handle circular arcs as well as lines; therefore, we only only need to extend the indexing table construction technique.

Extending the indexing table construction technique to generalized polygons involves formulating DB-curves and CB-curves for circular features in addition to linear features, as well as implementing routines for intersecting all combinations of these curves.

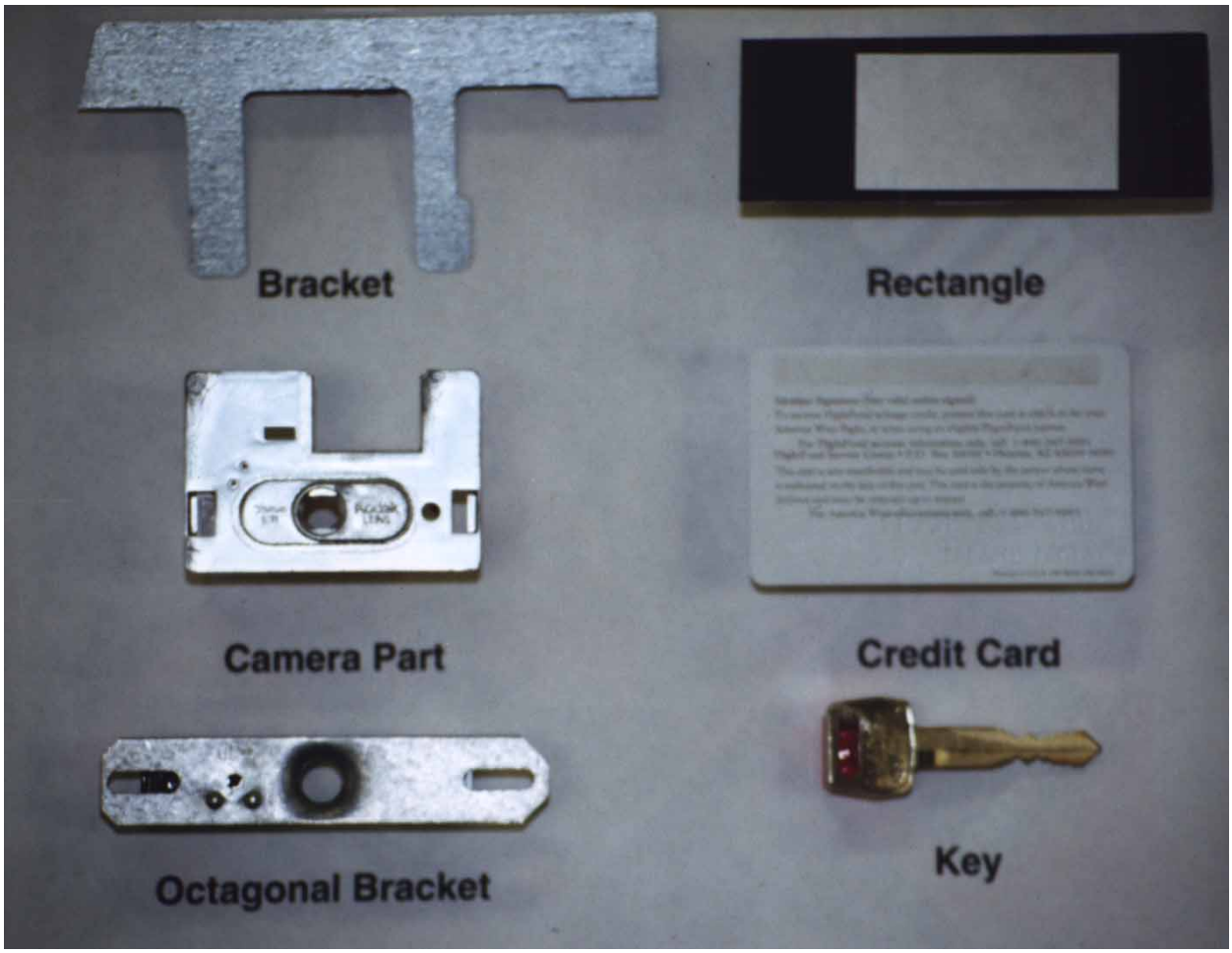


Figure 3.26: The set of possible objects.

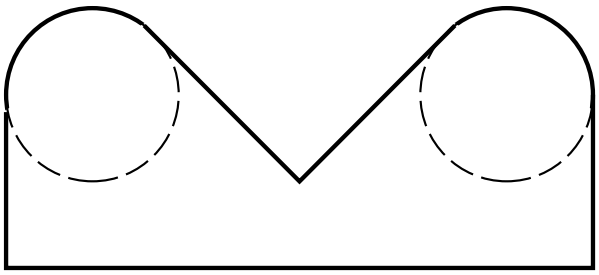


Figure 3.27: The boundary of a generalized polygon is composed of linear edges and circular arcs.

Object	Disc. (mm)	n	Table Size	κ	$\sigma(\kappa)$	Time (seconds)
Rectangle	2.5	4	13059	1.694	1.449	1041
Rectangle	1.5	4	29157	1.367	0.968	3328
Rectangle	1.0	4	59856	1.232	0.755	6366
Camera Part	2.5	6	39579	2.112	2.015	4666
Camera Part	1.5	6	83351	1.683	1.456	11133
Camera Part	1.0	6	161104	1.490	1.144	24013
Bracket	2.5	16	87297	1.553	1.107	34713
Bracket	1.5	16	218986	1.296	0.766	262601
Key	2.5	19	35416	3.054	2.526	5971
Key	1.5	19	119508	1.769	1.313	30956
Credit Card	2.5	4	26611	1.503	1.159	3020
Credit Card	1.5	4	66078	1.311	0.871	8354
Credit Card	1.0	4	140427	1.237	0.721	19287
Octagonal Bracket	2.5	8	61106	2.498	1.847	6632
Octagonal Bracket	2.0	8	85923	2.217	1.586	10344
Octagonal Bracket	1.5	8	135878	1.920	1.293	18572

Table 3.1: Sizes, average number of correspondences, standard deviations of the number of correspondences, and the construction time for complete indexing tables for various objects, resolutions.

The basic idea remains the same: we construct a complete indexing table by predicting indexing table entries for every pairwise intersection of DB-curves and CB-curves.

The key difference between the implementation for polygons and the implementation for generalized polygons is that we parameterize configuration space differently in order to simply the process of intersecting the curves. For polygonal models, we algebraically characterized DB-curves and CB-curves in terms of a (y, θ) configuration space. This approach has the advantage that it is general and all of the curves were defined with respect to the same variables, but for circular features, this approach is too inefficient, and a more complex approach is needed to achieve reasonable performance.

Expressing DB-curves and CB-curves for circular features would involve eighth order polynomial expressions in (y, θ) C-space. Furthermore, computing the intersections of two such curves would require solving 16^{th} order polynomials, which would be an expensive and inaccurate task.

For this reason, we parameterized DB-curves and CB-curves for circular features in such a way as to implicitly maintain contact between one characteristic point and one characteristic feature. This approach has the advantage that computing intersections between

DB-curves and CB-curves amounts to solving at most eighth order polynomial expressions. The disadvantage of this approach is that the curves are no longer defined in a uniform manner, *i.e.*, (y, θ), but rather, the curves are defined with respect to variables which depend upon the characteristic contact point and characteristic circular feature.

In section 3.5.2, we describe a parameterization (θ, ϕ) which implicitly achieves simultaneous contact between a point and a circular feature. In section 3.5.3, we formulate biquadratic expressions for DB-curves, and in section 3.5.4, we formulate biquadratic expressions for CB-Curves. In section 3.5.5, we describe a resultant-based technique for simultaneously solving pairs of biquadratic equations. In section 3.5.5, we step through the process of computing the intersections for two DB-Curves.

3.5.1 CB-Curves for Circular Arcs

In order to handle circular features, we must incorporate a new type of correspondence boundary curve which corresponds to a scanline tangentially contacting a circular arc, as shown in Figure 3.28. This type of contact is reducible to the CB-Curves we examined for polygonal objects where vertices contact scanlines. This reduction involves translating the scanline up or down by the circle's radius, and then, we are only concerned with the center of the circular arc contacting the scanline.

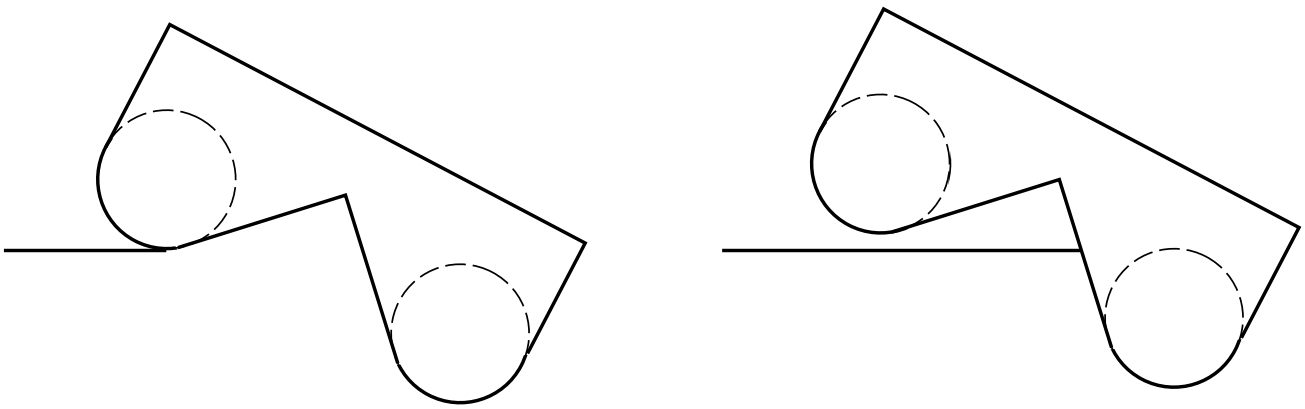


Figure 3.28: Correspondence changes can also result from the scanline contacting a circular arc.

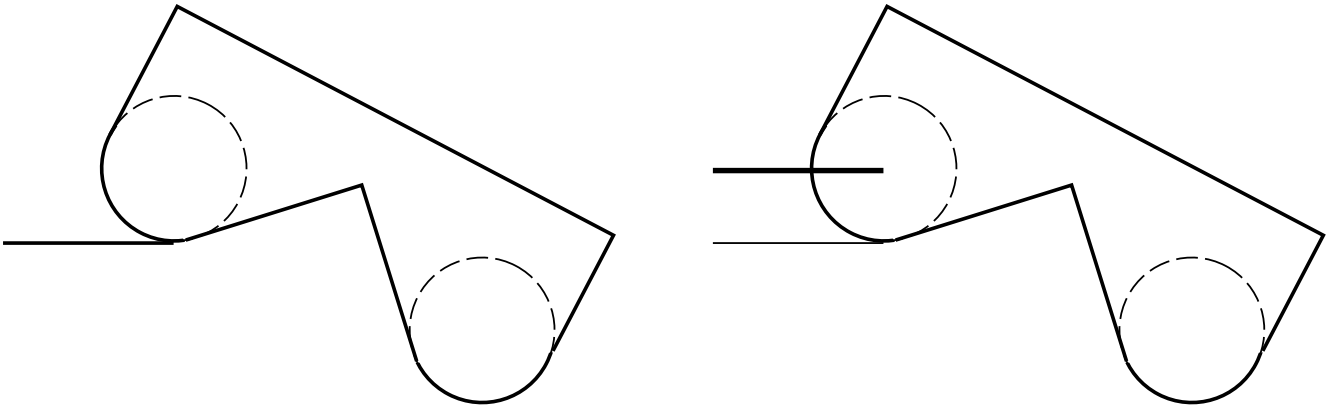


Figure 3.29: The correspondence boundary curves for circular arcs correspond to configurations where the scanline intersects the circle's center.

3.5.2 (θ, ϕ) Parameterization

In this section, we describe a pose parameterization which implicitly achieves contact between the characteristic point and the characteristic circular feature; we use this parameterization because it reduces the number of constraints and therefore simplifies the constraint formulations. The parameterization (θ, ϕ) corresponds to rotating the point set by θ and then translating the point set by $(R\cos(\phi), R\sin(\phi))$ (where R refers to the radius of the characteristic circle). This parameterization requires that both the characteristic point, and the center of the characteristic (parameterization) circle to lie coincidentally at the origin. The advantage of this parameterization is that we can compute the pose which satisfies three simultaneous constraints by reformulating the problem in terms of simultaneously satisfying two constraints because the first constraint is implicitly satisfied via the parameterization. Solving for the curve intersections via this parameterization involves three steps:

1. Translating the point set and feature set so that the characteristic point which corresponds to the characteristic circle, and the characteristic circle are both centered at origin

2. Express the two curves (CB-curves or DB-curves) in terms of biquadratic expressions in terms of (θ, ϕ)

3. Analytically compute the orientations θ corresponding to the intersections of these curves

4. Compute the translation (x, y) corresponding to each plausible orientation (θ)

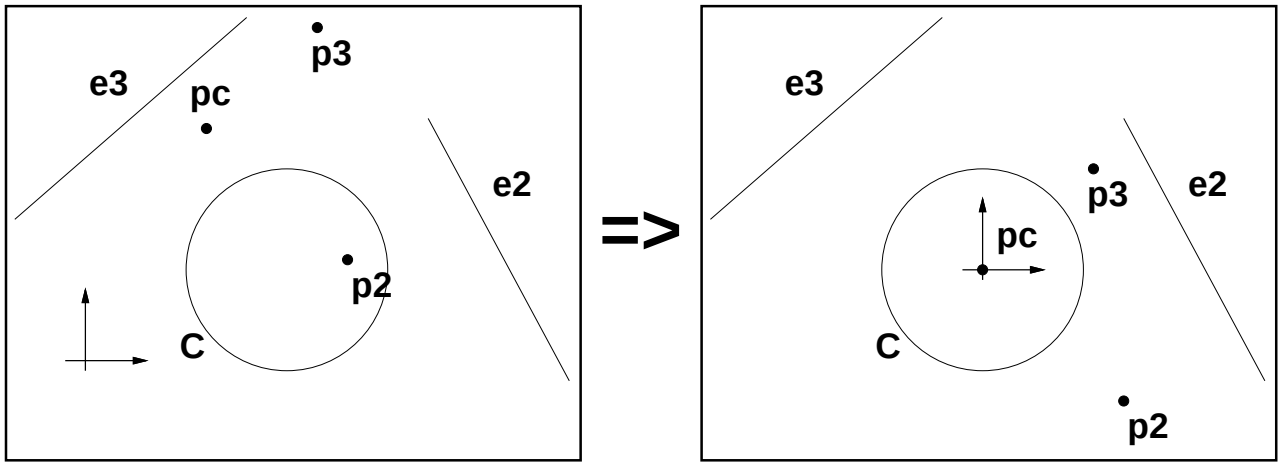


Figure 3.30: The parameterization requires that the the center of the characteristic (parameterization) circle and the characteristic point both lie at the origin in order for it to implicitly achieve contact between the characteristic point and the characteristic circular feature.

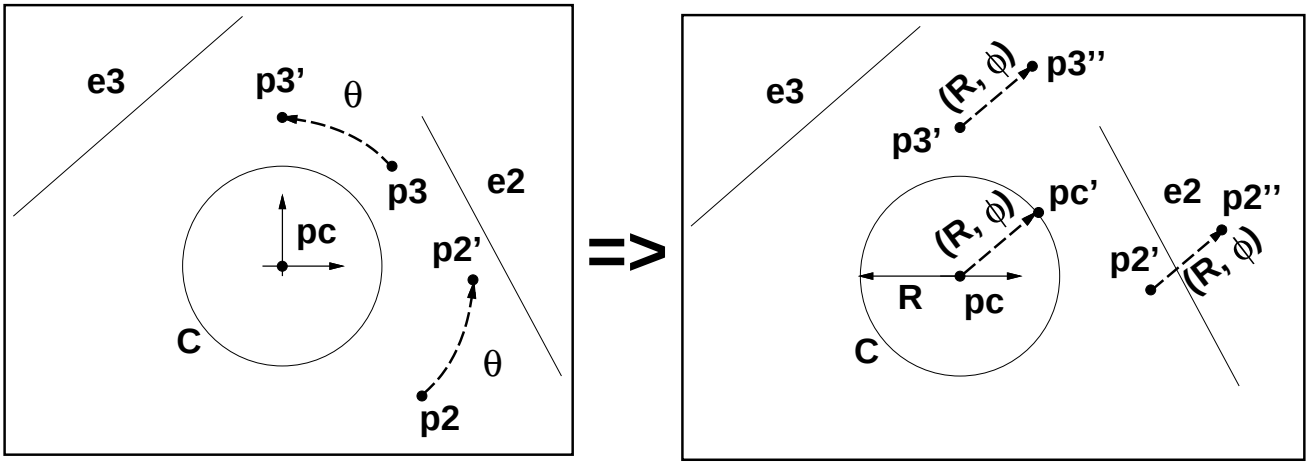


Figure 3.31: The parameterization involves rotating the other 2 points around the origin by θ and then translating them by the polar vector (R, ϕ).

Algebraic Formulation of the Parameterization Transform

Equation 3.6 presents the two-dimensional transformation, in homogeneous coordinates, corresponding to the (θ, ϕ) parameterization. Since utilizing algebraic elimination techniques requires algebraic expressions, we reformulate the transform via trigonometric substitutions $t = \tan(\frac{\theta}{2})$ and $u = \tan(\frac{\phi}{2})$.

$$\begin{bmatrix} \cos(\theta) & \Leftrightarrow\sin(\theta) & R\cos(\phi) \\ \sin(\theta) & \cos(\theta) & R\sin(\phi) \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} (1\Leftrightarrow t^2)(1+u^2) & \Leftrightarrow(2t)(1+u^2) & R(1+t^2)(1\Leftrightarrow u^2) \\ (2t)(1+u^2) & (1\Leftrightarrow t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \quad (3.6)$$

3.5.3 Algebraically Formulating Discretization Boundary Constraints

In this section, we formulate expressions for DB-Curve constraints which correspond to mapping a point onto a linear or circular feature.

Equation (3.7) presents an expression corresponding to transforming a point (X, Y) onto a line parameterized by (a, b, c) corresponding to $a\mathtt{X}+b\mathtt{Y} = c$. Actually, each linear constraint corresponds to curves of degree two, but the biquadratic form is more general; since linear constraint has fewer degrees of freedom than circular features, each linear constraint produces imaginary $\pm i$.

$$f(t,u) = \left(\begin{bmatrix} (1\Leftrightarrow t^2)(1+u^2) & \Leftrightarrow(2t)(1+u^2) & R(1+t^2)(1\Leftrightarrow u^2) \\ (2t)(1+u^2) & (1\Leftrightarrow t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \right)^T \cdot \begin{bmatrix} a \\ b \\ \Leftrightarrow c \end{bmatrix} \quad (3.7)$$

$$\begin{aligned} f(t,u) \;=\; & t^2(u^2(\Leftrightarrow c \Leftrightarrow aR \Leftrightarrow aX \Leftrightarrow bY) + 2bRu \Leftrightarrow c + aR \Leftrightarrow aX \Leftrightarrow bY) \\ & +t(1+u^2)(2bX \Leftrightarrow 2aY) + u^2(\Leftrightarrow c \Leftrightarrow aR + aX + bY) + 2bRu \Leftrightarrow c + aR + aX + bY \end{aligned}$$

Equation (3.8) is an expression corresponding to transforming a point (X, Y) onto a circle parameterized by (xc, yc, r) corresponding to $(xc\Leftrightarrow X)^2+(yc\Leftrightarrow Y)^2 = r^2$. Fortunately this bi-quartic expression is divisible by $(1+t^2)(1+u^2)$ producing a biquadratic expression (equation (3.9)).

$$K(t,u) \;=\; \left(\begin{bmatrix} (1-t^2)(1+u^2) & -(2t)(1+u^2) & R(1+t^2)(1-u^2) \\ (2t)(1+u^2) & (1-t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \right)$$

$$- \begin{bmatrix} xc \\ yc \\ (1+t^2)(1+u^2) \end{bmatrix}$$

$$g(t,u) = K(t,u)^T \cdot K(t,u) - (r(1+t^2)(1+u^2))^2 \tag{3.8}$$

$$\begin{aligned} g(t,u) = \frac{X(t,u)}{(1+t^2)(1+u^2)} = \\ & t^2(u^2(R^2 - r^2 + 2Rxc + xc^2 + 2RX + 2xcX + X^2 + yc^2 + 2ycY + Y^2) \\ & +u(-4Ryc - 4RY) + (R^2 - r^2 - 2Rxc + xc^2 - 2RX + 2xcX + X^2 + yc^2 + 2ycY + Y^2)) \\ & +t(u^2(-4Xyc + 4RY + 4xcY) + u(8RX) - 4Xyc - 4RY + 4xcY) \\ & +u^2(R^2 - r^2 + 2Rxc + xc^2 - 2RX - 2xcX + X^2 + yc^2 - 2ycY + Y^2) \\ & +u(-4Ryc + 4RY) + (R^2 - r^2 - 2Rxc + xc^2 + 2RX - 2xcX + X^2 + yc^2 - 2ycY + Y^2) \end{aligned} \tag{3.9}$$

3.5.4 Algebraically Formulating CB-Curves

In this section, we describe a technique for formulating correspondence boundary curves in terms of the (θ, ϕ) parameterization. These curves describe the situation where a feature point is transformed onto a horizontal scanline. It is important to note that DB-Curves involve transforming non-point model features onto points whereas CB-Curves involve transforming model vertices (points) onto horizontal scanlines (lines). In order to utilize a consistent framework for dealing with both cases (classifying points by (X, Y), lines by (a, b, c), and circles by (x_c, y_c, r)) , we make use of the inverse matrix of $R(\theta, \phi)$ (equation (3.10)) so that we can use a (θ, ϕ) parameterization for all of the curves, in order to simplify the task of intersecting the curves; this produces a consistent representation wherein all of the constraints correspond to mapping points onto non-point features: equation (3.7) expresses the CB-curve corresponding to transforming a line (X, Y) onto a point using the inverse transformation $R(\theta, \phi)^{-1}$.

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & R\cos(\phi) \\ \sin(\theta) & \cos(\theta) & R\sin(\phi) \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & -R(\cos(\theta)\cos(\phi) + \sin(\theta)\sin(\phi)) \\ -\sin(\theta) & \cos(\theta) & R(\cos(\theta)\sin(\phi) - \sin(\theta)\cos(\phi)) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$h(t,u)=\left(\begin{bmatrix} (1-t^2)(1+u^2) & (2t)(1+u^2) & -R((1-t^2)(1-u^2)+4tu) \\ -(2t)(1+u^2) & (1-t^2)(1+u^2) & -R((1-t^2)2u+2t(1-u^2)) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}\right)^T \cdot \begin{bmatrix} a \\ b \\ -c \end{bmatrix} \quad (3.11)$$

$$\begin{aligned} h(t,u) = {} & t^2(u^2(-c-aR-aX-bY)+u(2bR)+(-c+aR-aX-bY) \\ & +t(u^2(2bR-2bX+2aY)+u(-4aR)+(2bR-2bX+2aY)+ \\ & (u^2(-c+aR+aX+bY)+u(-2bR)+(-c-aR+aX+bY)) \end{aligned} \quad (3.12)$$

3.5.5 Solving Pairs of Biquadratic Equations

We determine the intersections of pairs of CB-curves and DB-curves by formulating the problem algebraically and using elimination techniques to solve for each variable one at a time. Elimination theory is an algebraic method for solving simultaneous systems of multivariate equations by constructing a particular equation in a single variable such that every root of that variable for all of the simultaneous solutions is also a root of the particular equation. Recently, Manocha [Man92] showed that the other coordinates of each root can be found in the intermediate computations. Still, the main drawback of this elimination approach is that the running time, the number of solutions, and numerical imprecision are all exponential in the number of variables and the polynomial degree of the expressions. For these reasons, it is desirable to formulate the constraints as compactly as possible to achieve the minimum algebraic complexity.

In this section, we step through the resultant method for simultaneously solving pairs of biquadratic equations. Assume $f_1(t,u)$ and $f_2(t,u)$ are algebraic expressions which are biquadratic in t and u. The Dixon resultant formulation [Dix08] of f_1 and f_2 provides a resultant matrix whose determinant is an equation solely in one of the variables, such that every value of that variable for all the simultaneous roots of $f_1(t,u) = f_2(t,u) = 0$ is a root of that resultant polynomial expression. The Dixon resultant construction involves separating $f_1(t,u)$ and $f_2(t,u)$ into monomial coefficients of u (or t respectively) (equations (3.13),(3.14)). Equation (3.15) presents the Dixon resultant matrix, and the resultant equation corresponds to $|M(t)| = 0$.

$$f_1(t,u) = f_{1,2}(t)u^2 + f_{1,1}(t)u + f_{1,0}(t) \quad (3.13)$$

$$f_2(t,u) = f_{2,2}(t)u^2 + f_{2,1}(t)u + f_{2,0}(t) \quad (3.14)$$

$$M(t) = \begin{bmatrix} f_{1,1}(t)f_{2,0}(t) \Leftrightarrow f_{1,0}(t)f_{2,1}(t) & f_{1,2}(t)f_{2,0}(t) \Leftrightarrow f_{1,0}(t)f_{2,2}(t) \\ f_{1,2}(t)f_{2,0}(t) \Leftrightarrow f_{1,0}(t)f_{2,2}(t) & f_{1,2}(t)f_{2,1}(t) \Leftrightarrow f_{1,1}(t)f_{2,2}(t) \end{bmatrix} \tag{3.15}$$

We could compute the roots of the resultant equation either by solving the matrix polynomial and transforming it into an eigenvalue problem [Man92], or we can compute the roots by expanding out the resultant matrix to a polynomial, and then solving it numerically by a companion matrix eigenvalue solver. The determinant of the resultant matrix expands into an eighth order polynomial. Even though the matrix polynomial formulation is inherently more accurate, we chose the latter approach because finding the roots of the univariate polynomial should provide sufficient accuracy.

$$|M(t)| = a_8t^8 + a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \tag{3.16}$$

Rather than constructing the resultant symbolically at runtime for every different pair of biquadratic equations, we can construct a generic resultant in terms of symbolic coefficients, and then instantiate these coefficients for each particular pair of biquadratic equations. Then we numerically solve for the roots of the expressions by constructing a companion matrix and computing the eigenvalues.

Example: Intersecting DB-Curves

In order to describe the process of computing the intersections of discretization boundary curves, we step through an example which involves computing the pose which achieves simultaneous contact between three points and two circles and a line.

In this example, point $A = (10, \Leftrightarrow 5)$ corresponds to circle **A** centered at $(5, 10)$ with radius 8. Point $B = (\Leftrightarrow 11, 5)$ corresponds to circle **B** centered at $(\Leftrightarrow 5, \Leftrightarrow 15)$ with radius 12. Point $C = (\Leftrightarrow 8, \Leftrightarrow 13)$ corresponds to the line **C** characterized by $x = 13 (a = 1, b = 0, c = 13)$. The correct transformation maps A onto $(5, 18)$, B onto $(\Leftrightarrow 5, \Leftrightarrow 3)$ and C onto $(13, 0)$.

First we translate the point and features sets so that the first point and the center of the first circle both lie at the origin (as shown in Figure 3.33). Point A is translated to $(0, 0)$, point B is translated to $(\Leftrightarrow 21, 10)$, and point C is translated to $(\Leftrightarrow 18, \Leftrightarrow 8)$. Circle **A** is translated to $(0, 0)$, circle **B** is translated to $(\Leftrightarrow 10, \Leftrightarrow 25)$ and line **C** is translated to $x = 8$.

Equation (3.17) corresponds to the constraint that point B is mapped onto circular feature **B**. Equation (3.18) corresponds to the constraint that point C is mapped onto linear

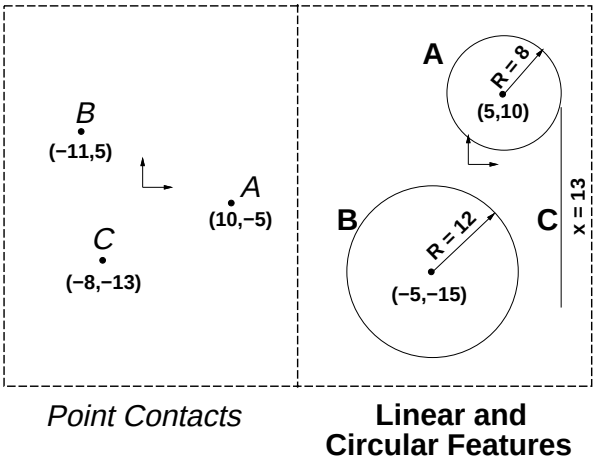


Figure 3.32: We step through an example for computing the pose which achieves simultaneous contact between three points.

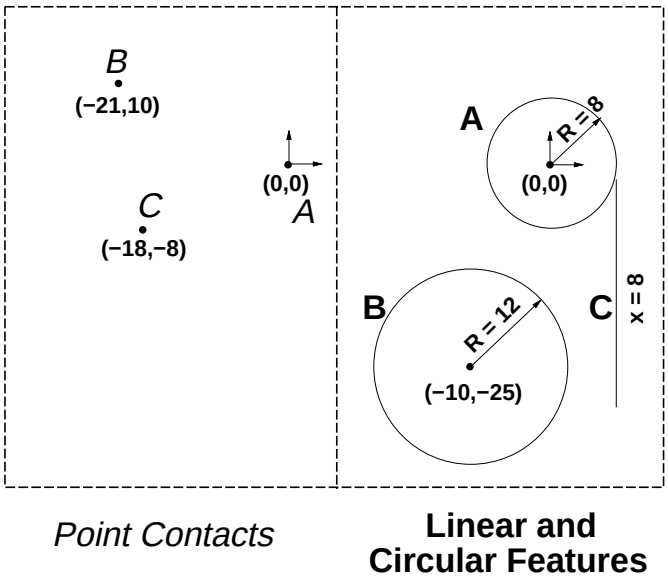


Figure 3.33: The first step is to transform the point sets and feature sets so that the first point and the center of the first circle both lie at the origin.

feature **C**. Next, we separate $f_1(t,u)$ and $f_2(t,u)$ into their u coefficients (equations (3.19)-(3.24)). Next, we compute the resultant matrix $M(t)$ (equation (3.25)).

$$f_1(t,u|R=8,X=\Leftrightarrow 21,Y=10,xc=\Leftrightarrow 10,yc=\Leftrightarrow 25,rc=12) = \tag{3.17}$$
$$1090 \Leftrightarrow 2820\,t + 1602\,t^2 + 1120\,u \Leftrightarrow 1344\,t\,u + 480\,t^2\,u + 1442\,u^2 \Leftrightarrow 2180\,t\,u^2 + 610\,t^2\,u^2$$
$$f_2(t,u|R=8,X=\Leftrightarrow 18,Y=\Leftrightarrow 8,a=1,b=0,c=8) = \tag{3.18}$$
$$\Leftrightarrow 18 + 16\,t + 18\,t^2 \Leftrightarrow 34\,u^2 + 16\,t\,u^2 + 2\,t^2\,u^2$$

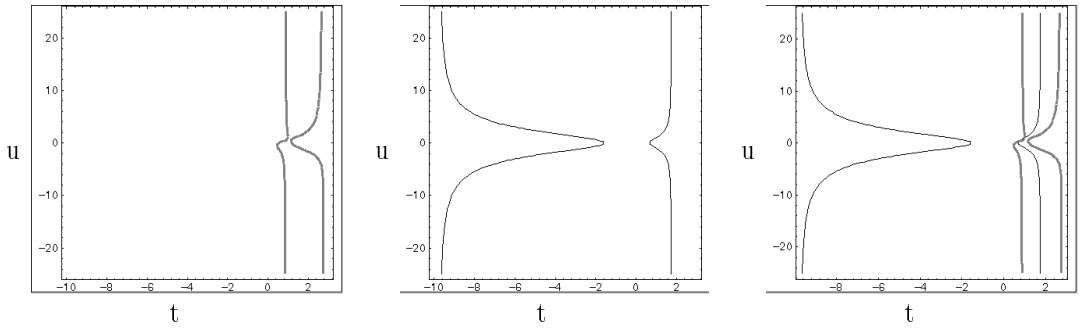
$$f_{1,0}(t) = 1602t^2 \Leftrightarrow 2820t + 1090 \tag{3.19}$$
$$f_{1,1}(t) = 480t^2 \Leftrightarrow 1344t + 1120 \tag{3.20}$$
$$f_{1,2}(t) = 610t^2 \Leftrightarrow 2180t + 1442 \tag{3.21}$$
$$f_{2,0}(t) = 18t^2 + 16t \Leftrightarrow 18 \tag{3.22}$$
$$f_{2,1}(t) = 0 \tag{3.23}$$
$$f_{2,2}(t) = 2t^2 + 16t \Leftrightarrow 34 \tag{3.24}$$

$$M(t) = \{\{-20160+42112\,t-9984\,t^2-16512\,t^3+8640\,t^4, 11104-51008\,t+77504\,t^2-49472\,t^3+7776\,t^4\},$$
$$\{11104-51008\,t+77504\,t^2-49472\,t^3+7776\,t^4, 38080-63616\,t+35584\,t^2-4992\,t^3-960\,t^4\}\} \tag{3.25}$$

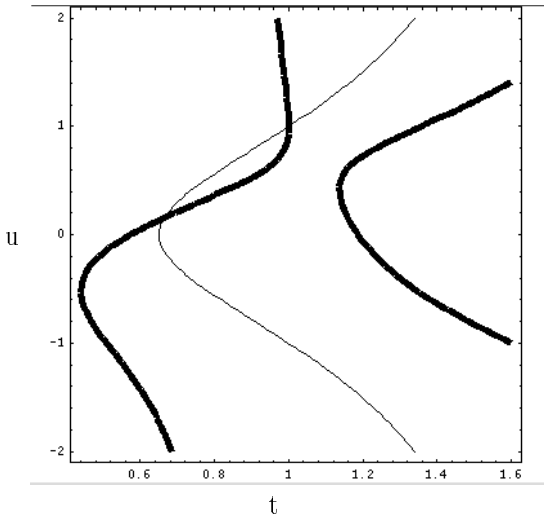
Next, we compute the resultant polynomial by taking the determinant of $M(t)$ (equation (3.26) and then solve for all values of t (equation (3.27)). The resultant polynomial has two real solutions : $t = 1$ and $t = 0.664139641$, and six complex solutions. The complex solutions do not correspond to poses achieving simultaneous contact; for complex t, $\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}$ correspond to scale transformations. Finally, we compute X_θ, Y_θ for both real solutions corresponding to $(X = \Leftrightarrow 8, Y = 0, \theta = \frac{-\pi}{2})$ and $(X = \Leftrightarrow 5.232, Y = 6.052, \theta = \Leftrightarrow 1.172)$ (Figure 3.35).

$$|M(t)| = -890991616 + 4018909184\,t - 8099586048\,t^2 + 10610839552\,t^3 - 10393196544\,t^4$$
$$+7334039552\,t^5 - 3253362688\,t^6 + 742109184\,t^7 - 68760576\,t^8 \tag{3.26}$$

$$t = \{1, i, \Leftrightarrow i, 0.664139641, 0.999468823 \Leftrightarrow 0.557401976i, 0.999468823 + 0.557401976i,$$
$$3.564788927 \Leftrightarrow 1.479918884i, 3.564788927 + 1.479918884i\} \tag{3.27}$$



f1(t,u) = 0 f2(t,u) = 0 f1(t,u) = 0 & f2(t,u) = 0



f1(t,u) = 0 & f2(t,u) = 0 (close up)

Figure 3.34: The roots of the expressions f1(t,u) = 0 & f2(t,u) = 0

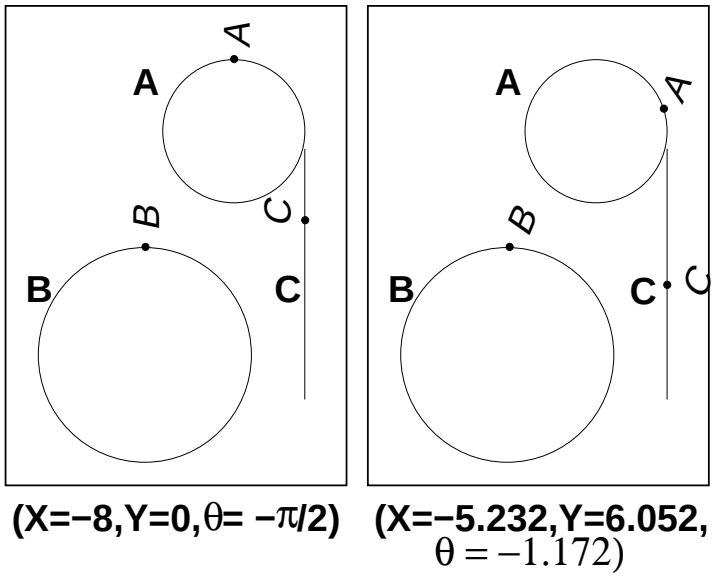


Figure 3.35: The two poses achieving simultaneous contact between the three points and the two circular features and one linear feature. In this figure, the points are transformed while the features stay in place.

3.5.6 Complete Lookup Table Sizes

We constructed complete indexing tables for more accurate generalized polygonal models of three of the objects shown in Figure 3.26, and for the objects shown in Figure 3.36. Table 3.2 lists the sizes of indexing tables, and κ, the average number of interpretations for each valid indexing coordinate, $\sigma(\kappa)$, the standard deviation of the number of interpretations, and the table construction time. The running times could have been decreased had we implemented either the $O(m\log(n))$ method or $O(m)$ method for synthesizing predicted indexing coordinates. Notice that the number of indexing table entries increases as one over the square of the discretization because we are dealing with a two degree of freedom system. We verified the completeness of the indexing tables by constructing finely discretized tables (2.5 mm) and generating data for 100 simulated scanning and validating that the table included the corresponding table entry.

Figure 3.36: Complete indexing tables were generated for various generalized polygonal objects.

Object	Disc. (mm)	n	Table Size	κ	$\sigma(\kappa)$	Time (seconds)
Camera Part	2.5	16	47131	2.617	2.557	8619
Camera Part	1.5	16	93564	1.962	1.725	18389
Camera Part	1.0	16	187053	1.633	1.286	145393
Bracket	2.5	25	109182	1.687	1.349	348912
Bracket	1.5	25	244079	1.396	0.935	714320
Credit Card	2.5	4	33552	2.023	2.180	13737
Credit Card	1.5	4	74749	1.577	1.417	31258
8 Handle	2.5	16	2740	3.321	2.009	895
8 Handle	1.0	16	5507	1.598	1.148	2145
Small 4 Handle	2.5	16	6782	3.321	2.579	15973
Small 4 Handle	1.0	16	22226	1.866	1.410	61995
Large 4 Handle	2.5	16	10072	2.607	2.075	25951
Large 4 Handle	1.0	16	36527	1.598	1.148	77155
Flange	2.5	4	19821	2.0171	2.031	16763
Flange	1.0	4	78017	1.445	1.265	70779
Triangular Part	2.5	4	20840	2.625	2.519	7803
Triangular Part	1.0	4	88481	1.667	1.426	31071

Table 3.2: Sizes, average number of correspondences, standard deviations of the number of correspondences, and the construction time for complete indexing tables for various objects, resolutions.

3.6 Localization Technique

In this section we outline an algorithm for estimating an object's pose given the sensed data and the correspondence interpretation. Most of the previous work in pose estimation deals with matching features of similar type, such as points to points, or lines to lines. For scanline data, scanline endpoints correspond to non-point features. The optimal pose estimate is computed using the localization algorithm described by Wallack and Manocha [WM94b] (Chapter 4)which computes the rigid two-dimensional transformation which optimally transforms the non-point features onto point features (Figure 2.8).

Their technique (which is described in Chapter 4) computes the transformation which minimizes the sum squared errors between the transformed points and the non-point features, where the error corresponds to the minimum distance between a point and a feature. The global minimum is found algebraically by formulating a sum squared error expression parameterized by (x, y, θ) between the point set and the set of non-point features. Local methods for computing the global minimum, such as Gauss-Newton's method, are susceptible to problems of local minima, and these problems can be exacerbated for small data sets. Wallack and Manocha's technique computes the global minimum by finding all the local extrema of the error function using a combination of numerical and algebraic techniques and then checking each of these extrema. All of the local extrema are found by solving a system of partial derivative equations using a combination of algebraic and numerical techniques to solve the system one variable at a time via resultants and matrix computations.

3.7 Experiment and Results

In this section, we discuss the results of experiments which analyzed the technique's recognition and localization performance. Positional and orientational repeatabilities were measured by localizing a precisely modeled rigidly-held object (a white rectangle imprinted on a piece of paper) multiple times along various scanning paths. This particular object was chosen because we wanted to focus upon the accuracy of the indexing technique rather than our modeling. The rectangle was printed using a laser printer and provided a precise, accurately modeled object for measuring the technique's performance. Recognition performance was measured by identifying known objects (silhouettes are shown in Figure 3.26)

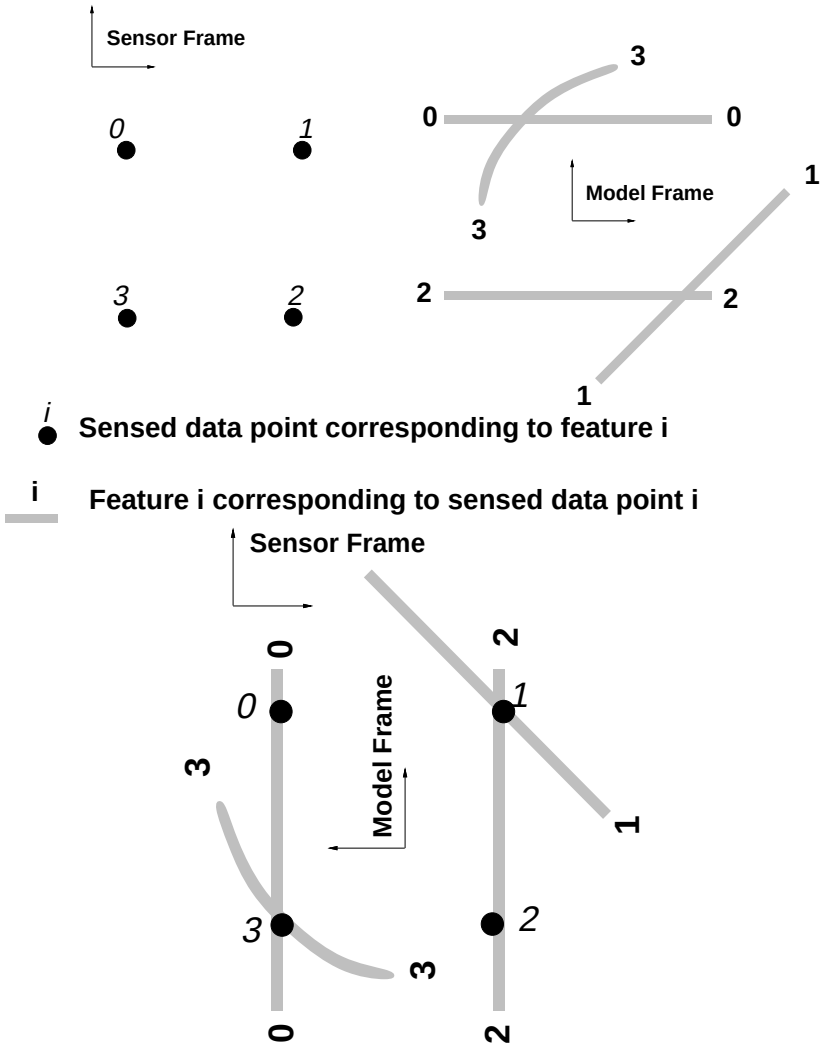


Figure 3.37: Given a set of sensed data points in the sensor reference frame, and non-point model features in a model reference frame (top), the task is to determine the model position, which is the transformation which makes the features converge (bottom).

in a variety of poses.

3.7.1 Positional Accuracy Results

The object was localized many times in order to determine the technique's localization repeatability. The object traversed many parallel paths 25 times each. The average localized positions, and standard deviations for all 25 localized positions are given in Table 3.3. The average localized position was computed for each path, and these averages agreed to within ± 0.01 mm. The standard deviations σ for each path were approximately 0.01 mm; a 2σ error bound assumption implies that localized position should vary by 0.02 mm.

path x-position (mm)	mean localized part position (mm)	std. dev. σ (mm)
400.00	(2.407, 3.270)	0.0061
410.00	(2.398, 3.276)	0.0067
420.00	(2.413, 3.278)	0.0077
430.00	(2.418, 3.276)	0.0077
440.00	(2.390, 3.286)	0.0111

Table 3.3: The mean localized positions, and the standard deviations of those (25) localized positions resulting from localizing a rectangle along different x ordinate paths.

3.7.2 Orientational Accuracy Results

The localization technique's orientational accuracy was measured by localizing the rectangle at known relative orientations. The differences between the end-effector's commanded orientation and the rectangle's computed orientation are given in Table 3.4. The difference between the commanded orientations and the average localized orientations differed by less than $\pm 0.03^o$.

3.7.3 Recognition Results

The objects shown in Figure 3.26 and the rectangles shown in Figure 3.38 were used to measure the scanning sensor's recognition performance by identifying each object 50 times in a variety of poses. The technique correctly identified each object every time.

mean robot orientation θ (^{o})	mean localized (^{o})	std. dev. σ (^{o})
72.0089	90.1853	0.0171
36.0076	90.2400	0.0224
-0.1017	90.1840	0.0233
-36.0649	90.2039	0.0266
-71.9008	90.2198	0.0270

Table 3.4: The commanded orientations, the mean estimated orientations, and the standard deviations of those (25) estimated orientations positions resulting from localizing a rectangle at different orientations.

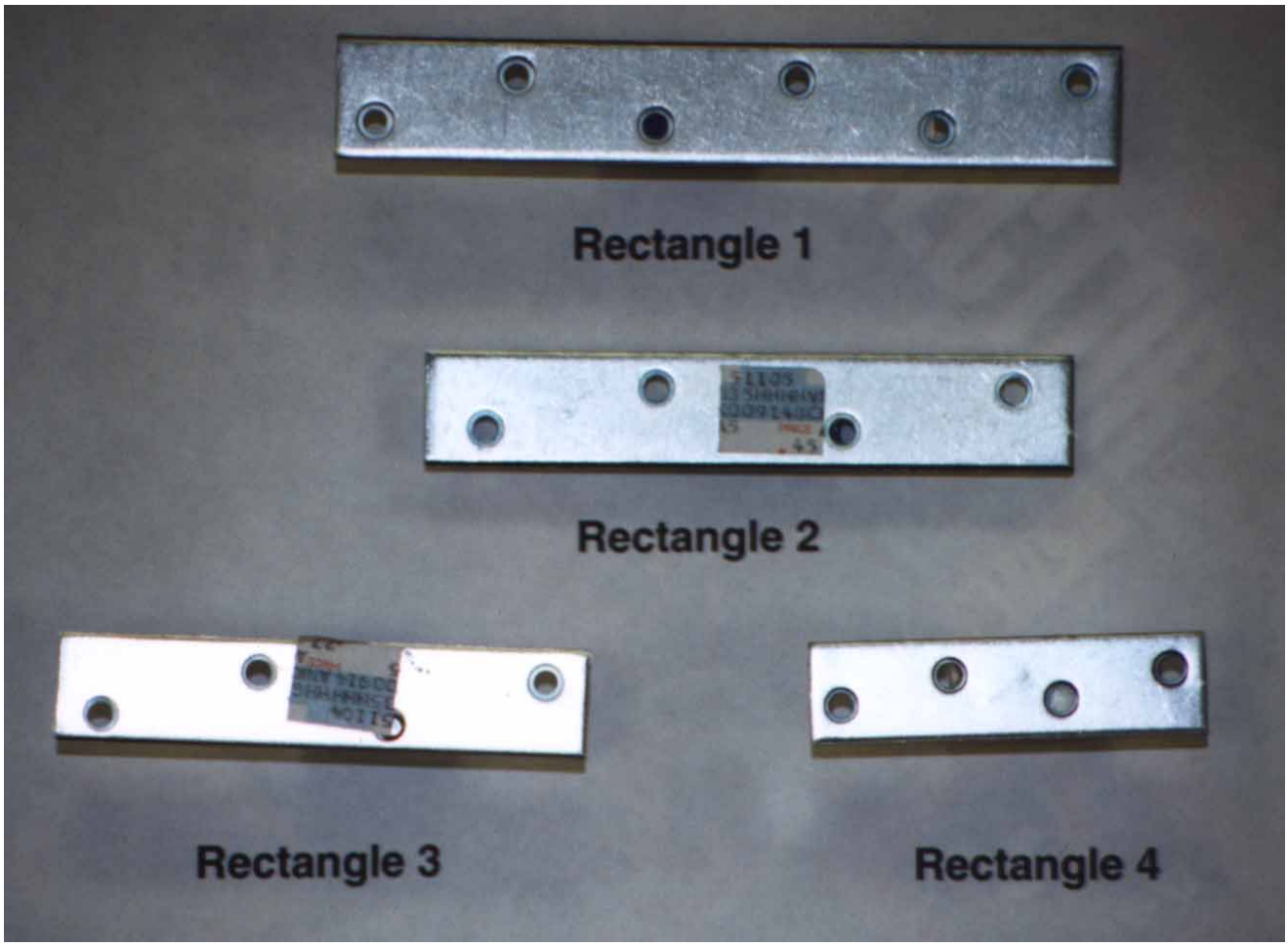


Figure 3.38: Four rectangles which were correctly identified 50 times each.

3.8 Conclusion

In this chapter, we presented a model-based object recognition and localization technique designed specifically for industrial manufacturing based on scanning beam sensors. Beam sensors are highly accurate, robust, fast, and inexpensive due to their simple specification. In order to utilize indexing to solve the correspondence problem in constant time, complete indexing tables were constructed. We described the method for constructing complete indexing tables and this method is applicable to generic systems. Objects were localized using a combination of algebraic and numerical methods to match the scanline endpoints to the non-point model features; positional estimates were repeatable to 0.025 mm, and orientational estimates were repeatable to 0.03^o.

Chapter 4

Robust Algorithms for Object Localization

Abstract

Object localization using sensed data features and corresponding model features is a fundamental problem in machine vision. We reformulate object localization as a least squares problem: the optimal pose estimate minimizes the squared error (discrepancy) between the sensed and predicted data. The resulting problem is non-linear and previous attempts to estimate the optimal pose using local methods such as gradient descent suffer from local minima and, at times, return incorrect results. In this chapter, we describe an exact, accurate and efficient algorithm based on resultants, linear algebra, and numerical analysis, for solving the nonlinear least squares problem associated with localizing two-dimensional objects given two-dimensional data. This work is aimed at tasks where the sensor features and the model features are of different types and where either the sensor features or model features are points. It is applicable to localizing modeled objects from image data, and estimates the pose using all of the pixels in the detected edges. The algorithm's running time depends mainly on the type of non-point features, and it also depends to a small extent on the number of features. On a SPARC 10, the algorithm takes a few microseconds for rectilinear features, a few milliseconds for linear features, and a few seconds for circular features.

4.1 Introduction

Model-based recognition and localization are fundamental tasks in analyzing and understanding two-dimensional and three-dimensional scenes [Hor89; Gri90]. There are many applications for vision systems: inspection and quality control, mobile robot navigation, monitoring and surveillance. The input to such systems typically consists of two-dimensional image data or range data. The model-based recognition problem is to determine which object O from a set of candidate objects $\{O_1, O_2, \ldots\}$ best explains the input data. Since the object O can have any orientation or location, the localization problem is to compute the pose p for a given object O which best fits the sensed data.

The problems of recognition and localization have been extensively studied in the vision literature [Hor89; Gri90]. The majority of the object recognition algorithms utilize edge-based data, where edges are extracted from image data using filter-based methods. Edge data is either comprised of pixels which have exceeded some threshold, or edge segments formed by straight line approximations of those pixels. On the other hand, objects are usually modeled as a set of non-point features, such as line segments, polygons, or parameterized curves. For this reason, localization algorithms must handle matching sensor features and model features which are of different types. We present an exact and robust model-based object localization algorithm for features of different types, where either the sensed features or the model features are points (refer Figure 4.1). Unlike Kalman filtering based approaches to the same problem, this algorithm is immune to problems of local minima, and unlike approaches which reduce the problem to matching point features which can be solved exactly [KSSS86; SS87; FH86], this algorithm does not introduce error by sampling points from model features.

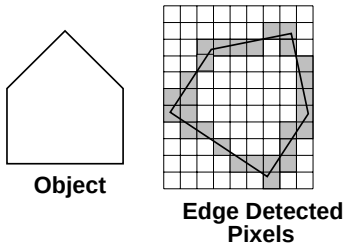


Figure 4.1: The localization algorithm estimates the object's pose directly from the edge detected pixels corresponding to model features.

Our approach involves reducing the model-based object localization problem be-

tween dissimilar features to a *nonlinear* least squares error problem, where the error of a pose is defined to be the discrepancy between the predicted sensor readings and the actual sensor readings. In this chapter, we concentrate on the object localization problem, *i.e.*, we assume that the features have already been interpreted using a correspondence algorithm. We use *global* methods to solve the problem of optimal pose estimation. In particular these methods compute the pose that minimizes the sum of the squared errors between the sensed data and the predicted data. Rather than using local methods such as gradient descent, the problem is formulated algebraically and all of the local extrema of the sum of squared error function, including the global minimum, are computed. For the case of matching points to circular features, we use an approximation of the squared error because the error cannot be formulated algebraically without introducing an additional variable for each circular feature.

The algorithm's running time mainly depends upon the algebraic complexity of the resulting system, which is a function the type of non-point features. Since the algorithm takes all of the sensor features and model features into account, the running time is linear in the size of the data set. It has been implemented and works well in practice. In particular, objects with rectilinear boundaries are localized in *microseconds* on a SPARC 10 workstation. The running time increases with the complexity of the non-point features: the algorithm takes a few *tens of milliseconds* to localize polygonal models, and one to two seconds to localize models with circular arcs.

4.1.1 Previous Work

The localization techniques presented in this chapter are applicable to the general field of model-based machine vision. In machine vision applications, localization is usually reformulated as a least squares error problem. Some methods have focused on interesting sensor features such as vertices, and ignored the remainder of the image data [GLP84; FH86; AF86; HU90; FM90; FMZ+91], thereby reducing the problem to a problem of matching similar features. These approaches are likely to be suboptimal because they utilize a subset of the data [Ros93]. In order to utilize all of the edge data, modeled objects must be compared directly to edge pixel data. In the model-based machine vision literature, there are three main approaches for localizing objects where the model features and the sensed features are of different types: sampling points from the non-point features to reduce it

to the problem of localizing point sets; constructing *virtual features* from the point feature data in order to reduce it to the problem of matching like features; estimating the object's pose by computing the error function between the dissimilar feature types and finding the minimum using gradient descent techniques, or Kalman filtering techniques.

One approach for object localization involves sampling points from non-point features in order to reduce it to the problem of matching points to points. The main advantage of reducing the problem of matching points to non-point features to the problem of matching point sets is that the problem is exactly solvable in constant time. For two-dimensional rigid transformations, there are only three degrees of freedom: x, y, θ [KSSS86; SS87]. By decoupling these degrees of freedom, the resultant problems become linear, and are exactly solvable in constant time. The crucial observation was that the relative position, (x, y), of the point sets could be determined exactly irrespective of the orientation, θ since the average of the x-coordinates and y-coordinates was invariant to rotation. This decoupling technique is valid for three dimensions as well [FH86]. To reduce the problem of matching points to non-point features to the solvable problem of matching points to points, points are sampled from the non-point features. This approach is optimal when objects are modeled as point-sets, but can be suboptimal if objects are modeled as non-point features, because sampling introduces errors when the sampled model points are *out of phase* with the actual sensed points.

Another approach to object localization for dissimilar features involves constructing *virtual features* from sensed feature data and then computing the object's pose by matching the like model features and *virtual features*. Such approaches solve an artificial problem, that of minimizing the error between *virtual* features and model features rather than minimizing the error between the sensed data and the modeled objects. Faugeras and Hebert [FH86] presented an algorithm for localizing three-dimensional objects using pointwise positional information. In their approach, virtual features were constructed from sensed data and then, objects were identified and localized from the features, rather than the sensed data. Forsyth *et al.* described a three-dimensional localization technique which represented the data by parameterized conic sections [FMZ+91]. Objects were recognized and localized using the algebraic parameters characterizing the conic sections. Recently, Ponce, Hoogs, and Kriegman have presented algorithms to compute the pose of curved three-dimensional objects [PHK92; PK92]. They relate image observables to models, reformulate the problem geometrically, and utilize algebraic elimination theory to solve a least

squares error problem. One drawback of their approach is that they minimize the resultant equation, not the actual error function. Ponce *et al.* use the Levenberg-Marquardt algorithm for the least square solution, which may return a local minimum [PHK92].

Another approach to object localization involves reducing it to the correct least squares problem, but then solving that least squares problem using gradient descent techniques. The least squares approach involves formulating an equation to be minimized; consequently, the error function between a point and a feature segment cannot take into account feature boundaries. Gradient descent methods require an initial search point and suffer from problems of local minima. Huttenlocher and Ullman used gradient descent techniques to improve the pose estimates developed from comparing sets of three model point features and three image point features [HU90]. Many localization algorithms have relied on Kalman filters to match points to non-point features. Kalman filters suffer from local minima problems as well. Given a set of points and corresponding non-point features, Ayache and Faugeras [AF86] attempted to compute the least squares error pose by iteration and by Kalman filters. Kalman filter-based approaches have demonstrated success in various machine-vision minimization applications [KDN93; ZF90; DF90]. Kalman filter-based approaches suffer from two problems: sensitivity to local minima problems, and the requirement of multiple data points.

A great deal of work has been done on matching features (points to points, lines to lines, points to features etc.) in the context of interpreting *structure from motion* [FM90; Hor91]. The problem involves matching sensor features from two distinct two-dimensional views of an object; the problem of motion computation is reduced to solving systems of non-linear algebraic equations. In practice, the resulting system of equations is overconstrained and the coefficients of the equations are inaccurate due to noise. As a result, the optimal solution is usually found using gradient descent methods on related least squares problems. Most of the local methods known in the literature do not work well in practice [JJ90] and an algorithm based on global minimization is greatly desired.

Sparse sensors have recently demonstrated success in recognition and localization applications [WCM93a; PC93]. The need for accurate localization algorithms has arisen with the emergence of these sparse probing sensor techniques. This technique has been applied to recognition and localization using beam sensor *probe data* and results in pose estimation with positional accuracy of $\frac{1}{1000}$" and orientational accuracy of 0.3^o [WCM93a; WCM93b].

The algorithm is applicable to both sparse and dense sensing paradigms; in sparse sensing applications, such as probing, a small set of data is collected in each experiment, whereas in dense sensing applications, such as machine vision, an inordinate amount of data is collected in each experiment. Sparse sensing applications necessitated development of this localization algorithm because sparse sensing strategies require an exact optimal pose strategy. First, a small data set cannot accommodate sampling points from the data set in order to transform the problem of matching features of dissimilar types to matching features of similar types, and second, a small data set is more susceptible to problems of local minima which can arise with local methods such as gradient descent or Kalman filtering.

4.1.2 Organization

The rest of the chapter is organized in the following manner. In Section 2, we review the algebraic methods used in this chapter and explain how probabilistic analysis provides a basis for using a least square error model to solve the localization problem. We describe the localization problem and show how algebraic methods can be applied for optimal pose determination in Section 3. The algorithm for localizing polygonal objects given point data is described in Section 4 and the algorithm for localizing generalized polygonal objects, *i.e.*, objects with linear and circular features is described in Section 5. In Section 6, we present various examples, and use the localization technique to compare the relative localization accuracy of utilizing all available data compared to using only an *interesting* subset of the data. Finally, we conclude by highlighting the contributions of this work.

4.2 Background

4.2.1 Error Model

In this section, we describe the error model used for localization. Probability theory provides the basis or reducing the object localization to a least squares error problem [Sch79]. Assuming imperfect sensing and imperfect models, our goal is to tolerate these discrepancies and provide the best estimate of the object's pose. Real data presents constraints which are inconsistent with perfect models, and for this reason, we reformulate

the problem into a related solvable problem; minimizing the squared error is a common method for *solving* problems with inconsistent constraints, and it provides to the maximum likelihood estimate, assuming that the errors (noise) approximate independent normal (Gaussian) distributions.

The reason we assume a normal (Gaussian) error distribution is the composition of error from many different sources. Even though we may be using high precision, high accuracy sensors to probe a two-dimensional object, such sensors have many intrinsic sources of error. Furthermore, real objects can never be perfectly modeled, and this accounts for an additional source of error. These error distributions define a conditional probability distribution on the sensed data values given the boundary position (which depends on the object's pose, equation (4.1)). The conditional probability can be rewritten in terms of the sum of the squared discrepancies (equation (4.2)). The conditional probability of the pose given the sensor data is related to the conditional probability of the sensor data given the pose by Bayes' theorem (equations (4.3,4.4)).

The minimum sum squared error pose corresponds to the maximum probability (maximum likelihood estimate), since minimizing the negated exponent maximizes the result of the whole expression. Thereby, computing the least squares error pose is analogous to performing maximum likelihood estimation.

$$\Pr(sensor = S|pose = \Theta) \propto e^{-\frac{Discrepancy(s_1,s_{1,predicted})^2}{2\sigma^2} - \frac{Discrepancy(s_2,s_{2,predicted})^2}{2\sigma^2} - \cdots} \quad (4.1)$$

$$\Pr(sensor = S|pose = \Theta) \propto e^{-\frac{\sum_i Discrepancy(s_i,s_{i,predicted})^2}{2\sigma^2}} \quad (4.2)$$

$$\Pr(pose = \Theta|sensor = S) = \frac{\Pr(sensor = S|pose = \Theta)\Pr(pose = \Theta)}{\Pr(sensor = S)} \quad (4.3)$$

$$\Pr(pose = \Theta|sensor = S) \propto e^{-\frac{\sum_i Discrepancy(s_i,s_{i,predicted})^2}{2\sigma^2}} \frac{\Pr(pose = \Theta)}{\Pr(sensor = S)}$$

$$\Pr(pose = \Theta|sensor = S) \propto e^{-\frac{\sum_i Discrepancy(s_i,s_{i,predicted})^2}{2\sigma^2}} \frac{1}{2\pi} \quad (4.4)$$

4.2.2 Directly Comparing Features

The sensor data point features should be directly compared to the model features, even when the sensor features and model features are of different types. One measure of discrepancy between points and nonpoint features is the Euclidean distance between a sensor data point and a specific predicted point on the model feature. Measuring the discrepancy

in this way relies on the mistaken assumption that the sensed point feature corresponds to a known point on the model feature. The correspondence information specifies a model feature relating to each sensor feature point, it does not indicate particular points on the model features. A more reasonable measure of the discrepancy is the minimum distance between the sensor feature point and the corresponding model feature, such as the minimum distance between a point and a line, or the minimum distance between a point and a circle. For machine vision applications, measuring the error in this way allows one to utilize all of the edge detected pixels without introducing any artificial error (by sampling the model features, for example). It turns out that the discrepancy (error), or minimum distance, between a sensor data point and a linear model feature or a circular arc model is a nonlinear computation. In particular, the minimum distance between a point and a circular arc cannot be formulated exactly without introducing an additional variable; in order to achieve constant time performance, we use a approximation to that minimum distance function which does not require introduction of an additional variable.

4.2.3 Solving Algebraic Systems

The key to this algorithm is that we reduce the problem of finding the global minimum of an algebraic function to solving a system of nonlinear algebraic equations. Finding the global minimum of a function is a classic problem and there many algorithms known in the literature. The algorithms can be categorized into local methods and global methods. Local techniques such as Newton's method require good initial estimates in order to find all the solutions in the given domain, and this is difficult in our applications. Global methods such as resultants and Gröbner bases are based on symbolic reduction and computing roots of univariate polynomials. They are rather slow in practice and in the context of floating point computations suffer from numerical problems. Other global techniques include homotopy methods. However, they are rather slow for practical applications [Hor91]. In this chapter, we use some recently developed algorithms based on multipolynomial resultants and matrix computations described by Manocha [Man92]. Resultants are well known in classical algebraic geometry literature [Sal85; Mac02] and have recently been effectively used in many problems in robotics, computer graphics and geometric modeling [Man92].

The algorithm for solving non-linear equations linearizes the problem using resultants and uses matrix computations such as eigendecomposition, singular value decom-

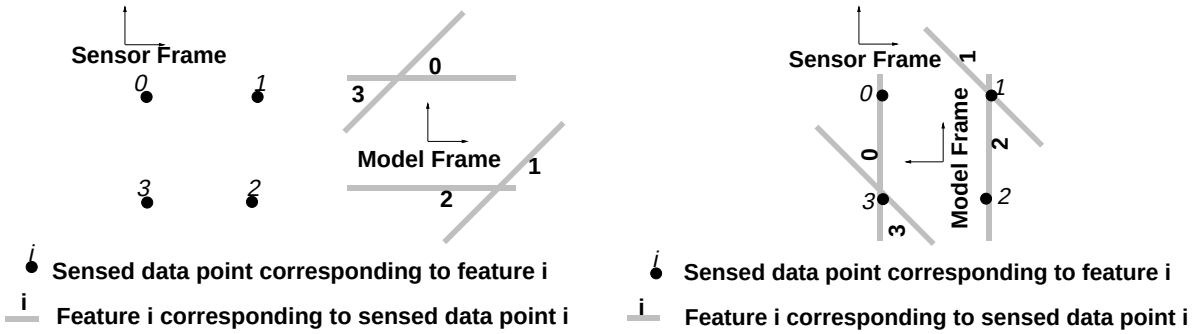


Figure 4.2: (a) Given a set of sensed data points in the sensor reference frame, and non-point model features in a model reference frame, the task (b) is to determine the model position (in the sensor reference frame).

position and Gaussian elimination [GL89]. Routines for these matrix computations are available as part of standard numerical linear algebra libraries. In particular, we used LAPACK [ABB+92] and EISPACK [GBDM77] routines in our implementation of the algorithm.

4.2.4 Problem Specification

Given a set of point features in one frame of reference and non-point features defined in another frame of reference, we determine the transformation which optimally maps the features onto each other. Assume without loss of generality that sensor features are data points (in the sensor reference frame) and correspond to non-point model features (in the model frame). The task is to localize the model (object) in the sensor reference frame; i.e., to determine the transformation between the sensor frame and the model frame consistent with the data (see Figure 4.2(a)). Figure 4.2(b) shows the most likely pose of a particular model/object for the points shown in Figure 4.2(a).

There are two approaches we can take: transform the non-point model features, and then match them to points, or transform the points, and then match them to the non-point model features. We have chosen to transform the points because it enables us to use a uniform approach to solving the mathematical problems corresponding to different feature types. Due to the duality of these two related transformations, the algorithm for solving for the optimal transformation of the points can also be used to solve for the optimal transformation of the non-point model features by computing the inverse transformation.

4.3 Theoretical Framework

4.3.1 Transformation Matrices

Two-dimensional rigid transformations can be parameterized by X, Y, θ (refer equation (4.5)). In order to arrive at algebraic equations, we utilize the trigonometric substitution ($t = \tan(\frac{\theta}{2})$: $\sin\theta = \frac{2t}{1+t^2}$ and $\cos\theta = \frac{1-t^2}{1+t^2}$). The algebraic matrix $Mat(X, Y, t)$, which is used throughout this chapter, is given in equation (4.5). It is important to remember that $Mat(X, Y, t)$ is not exactly $\mathcal{T}(X, Y, \theta)$ but rather $(1 + t^2)\mathcal{T}(X, Y, \theta)$.

$$\mathcal{T}(X,Y,\theta) = \begin{bmatrix} \cos(\theta) & \Leftrightarrow\sin(\theta) & X \\ \sin(\theta) & \cos(\theta) & Y \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow Mat(X,Y,t) \overset{t=\tan(\frac{\theta}{2})}{=} \begin{bmatrix} 1\Leftrightarrow t^2 & \Leftrightarrow 2t & X(1+t^2) \\ 2t & 1\Leftrightarrow t^2 & Y(1+t^2) \\ 0 & 0 & 1+t^2 \end{bmatrix} \tag{4.5}$$

4.3.2 Determining Local Extrema

One way to extract the global minimum of a function is to compute all of the local extrema (by solving for $\vec{x}$ in $\nabla \cdot F(x_1, x_2, \ldots, x_n) = (0, 0, \vec{\ldots}, 0)$) and find the minimum function value at all the local extrema. This approach succeeds in the non-degenerate case, when the global minimum may correspond to one or more points (a zero dimensional set), and when there are a finite number of local extrema, such as in the case of object localization. We utilize this basic approach in order to find the minimum squared error pose. Let $Error(X, Y, t)$ describe the total squared error as a function of the pose parameters. First, we find all the local extremum of $Error(X, Y, t)$ by solving $\nabla \cdot Error(X, Y, t) = (0, \vec{0}, 0)$. Then, the global minimum is found by comparing the error function at all of the local extrema poses.

Previous attempts to compute the global minimum of the error function using gradient descent methods suffer from problems of local minima because the partial derivatives of the error function are nonlinear functions. In this algorithm, we utilize elimination techniques to solve for all of the roots of the multivariate system of equations and thus avoid the problems of local methods.

4.3.3 Symbolic Representation

Elimination theory deals with solving for coordinates of simultaneous solutions of multivariable systems [Mac02]. Resultant methods are used for projection and elimination. We use resultants to solve for one particular coordinate at a time; these methods involve constructing a *resultant equation* in each coordinate, where the resultant construction depends on the structure of the underlying system of equations.

The resulting computation involves a considerable amount of symbolic preprocessing. As opposed to constructing the resultant formulation for each data set, we use a generic formulation. The algebraic formulation of the squared error between the transformed points and the lines allows us to use the algebraic equation as a generic representation for the squared error. The total error function between points and linear features can be expressed as a rational polynomial, whose numerator consists of 24 monomials terms. Likewise, the algebraic formulation of the squared approximate error between transformed points and circles allows us to use the algebraic equation as a generic representation for the squared error. The approximate total error function between points and linear features and circular features has a symbolic representation as a rational polynomial with a numerator consisting of 50 monomial terms. The algebraic formulation of the total error functions serves as the intermediate description. After reparameterizing the data sets in terms of algebraic coefficients, only one resultant formulation is necessary.

For each set of points and features, the sum of squared error between the set of points and corresponding features is described by a rational equation in terms of the pose parameters X, Y, t. The numerator of this rational equation is describable by the algebraic coefficients $X^iY^jt^k$, where our nomenclature is such that $\mathtt{ax_iy_jt_k}$ refers to the coefficient for the monomial $X^iY^jt^k$.

The multipolynomial resultant is used to eliminate variables from the system of partial derivative equations, and thus to produce a function in one variable. This univariate function is necessarily zero for all values of that variable which are common roots to the system of equations. In the context of this application, the orientation parameter t is the most difficult pose parameter to determine. The first step in both of these techniques involves *eliminating* X, Y to produce a resultant polynomial in t for which the partial derivatives of the error function are simultaneously zero. For each such t, we can then compute the two remaining pose parameters X_t and Y_t. In the case of polygonal models, X_t

and Y_t are determined by solving a system of two linear equations in X_t and Y_t. For models with circular arc features, computing $X_{Y_t,t}$ and Y_t involves setting up another resultant to compute Y_t by eliminating X_t, and then computing $X_{Y_t,t}$.

4.4 Optimal Pose Determination for Models With Linear Features

In this section, we describe the algebraic localization technique for polygonal models given the boundary data points. We define the problem as follows: compute the two-dimensional rigid transformation $\mathcal{T}(X, Y, \theta)$ which best transforms the linear model features $\vec{a}$ onto the data points $\vec{x}$ (refer section 4.2.4). This is accomplished by finding the two-dimensional rigid transformation $\mathcal{T}(X, Y, \theta)$ which best transforms the data points $\vec{x}$ onto the linear model features $\vec{a}$ and then compute the inverse transformation. Pose estimation is reduced to a least squares error problem, which is accomplished by computing all of the local extrema of the error function, and then checking the error function at all of the local extrema.

The main distinction between our approach and the related work in the literature lies in our use of algebraic elimination theory to solve the simultaneous system of partial derivative equations. We believe that gradient descent methods are unsuitable for solving this system of equations. Algebraic elimination methods are used for the principal step of solving the system of equations to enumerate the poses with local extrema of the error function.

The rest of the section is organized in the following manner. We first present an overview of the algorithm. Then, we present an algebraic formulation of the error between an arbitrary transformed point and an arbitrary line, followed by a generic description of the total error function, in terms of symbolic algebraic coefficients. Then, we detail the symbolic resultant matrix for solving for one of the pose parameters (t) of the generic error function.

The symbolic coefficients are recomputed each time the algorithm is run. In section 4.4.3, we describe numerical methods for computing all the orientations t corresponding to the local extrema poses. In section 4.4.4, we describe how the remaining pose parameters X_t, Y_t are computed for each orientation t.

Throughout this section, we will try to elucidate the main concepts using a simple example with four points corresponding to three lines. We will graphically depict the error function between each point and each corresponding linear feature, and we will depict the sum squared error function between the points and linear features. Unfortunately though, since the problem has three degrees of freedom, x, y, θ, and we want to use an additional degree of freedom to describe the error function, we cannot graphically depict the entire problem completely since we do not want to confuse the situation by trying to describe a four dimensional scenario. For these reasons, the graphics depict the three dimensional slice along the hyperplane $x = 0$.

We run through an example in section 4.4.5, and in section 4.4.7 we present a data set for which gradient descent methods may fail to find the optimal pose estimate. Finally, we present a faster localization technique for rectilinear objects which exploits a symbolic factoring of the resultant matrix.

4.4.1 Algorithmic Overview

Algebraic methods are used to compute the orientation, θ, of polygonal objects. This involves online and offline computations: the symbolic resultant matrix for θ is constructed from the symbolic partial derivatives offline; online, the symbolic coefficients are computed given the data set, thereby constructing the symbolic resultant matrix. Then, the resultant matrix is *solved* to enumerate all of the *orientations*, θ, of poses with local extrema of the error function. Then, the remaining pose parameters (X, Y) are determined for each orientation θ. Finally, the global minimum is found by comparing the error of all of the local extrema poses.

The localization algorithm involves four offline steps:

1. Determine the structure of a generic error function by examining an algebraic expression of the error between an arbitrary transformed point and an arbitrary linear feature.

2. Express the error function $Error(X, Y, \theta)$ as a generic algebraic function in terms of symbolic coefficients.

3. Formulate the partial derivatives of the generic error function $Error(X, Y, \theta)$ with respect to the coordinates: X, Y, and θ. The motivation is that each zero-dimensional

pose with local extremum error satisfies $\nabla \cdot Error(X, Y, \theta) = \vec{0}$.

4. Eliminate X and Y from the system of partial derivatives $\nabla \cdot Error(X, Y, \theta) = \vec{0}$ in order to generate a expression *solely in* θ. The result of this step is a symbolic resultant matrix which will be used to solve for all of the θ of poses with local extrema error function. Given the orientation, θ, the remaining pose parameters X_θ, Y_θ can be determined by solving a system of linear equations. .

In addition to the offline steps, the localization algorithm involves three online steps (given a data set):

1. Instantiate the symbolic coefficients of the error function using the data set of points and associated linear features.

2. Compute all of the candidate orientation parameters θ by solving the resultant matrix polynomial using eigenvalue techniques (refer section 4.4.3)

3. $\forall\ \hat{\theta}$, if $\hat{\theta}$ is a candidate orientation ($\mathrm{Im}(\hat{\theta}) \approx 0$) (where $Im(x)$ refers to the imaginary component of a complex number x):

   (a) Compute the remaining pose parameters X_θ, Y_θ for each θ by solving the linear system of equations $\frac{\partial Error(X,Y,\theta)}{\partial X}|_{\hat{\theta}} = 0$, $\frac{\partial Error(X,Y,\theta)}{\partial Y}|_{\hat{\theta}} = 0$

   (b) Compute $Error(X_{\hat{\theta}}, Y_{\hat{\theta}}, \hat{\theta})$ at each local extremum in order to find the global minimum error pose.

4.4.2 Points and Linear Features

In this section we highlight the squared error function between points and linear features. Given a point $(\mathtt{X},\mathtt{Y})$ and an associated linear feature described by $\vec{a} = (\mathtt{a},\mathtt{b},\mathtt{c})$ where $\mathtt{a}X + \mathtt{b}Y = \mathtt{c}$, the error between the point and the feature is: $\mathtt{aX + bY - c}$, or $(\mathtt{a},\mathtt{b},\Leftrightarrow\mathtt{c}) \cdot (\mathtt{X},\mathtt{Y},\mathtt{1})$. Let $\vec{a}$ be $(\mathtt{a},\mathtt{b},\Leftrightarrow\mathtt{c})^{\mathsf{T}}$ and $\vec{x}$ be $(\mathtt{X},\mathtt{Y},\mathtt{1})$. The squared error between the point $\vec{x}$ and the feature $\vec{a}$, as a function of the transformation $\mathcal{T}(X, Y, \theta)$ is given in equation (4.6). The total squared error for all of the points and associated features is given in equation (4.7). In order to realize rational functions, we use the trigonometric substitution $t = \tan(\frac{\theta}{2})$ to arrive at equation (4.7).

To circumvent the problems related to numerical inaccuracy, the points and features in the data set should be centered at the origin before invoking the algorithm, and the

optimal transformation should be complementarily transformed afterwards. This reduces the number of symbolic coefficients and thus improves the numerical precision.

$$\|(\mathtt{a},\mathtt{b},\Leftrightarrow\mathtt{c})^{\mathsf{T}}\cdot(\mathcal{T}(X,Y,\theta)(\mathtt{X},\mathtt{Y},\mathtt{1})^{\mathsf{T}})\|^2=\frac{\|(\mathtt{a},\mathtt{b},\Leftrightarrow\mathtt{c})^{\mathsf{T}}\cdot(Mat(X,Y,t)(\mathtt{X},\mathtt{Y},\mathtt{1})^{\mathsf{T}})\|^2}{(1+t^2)^2} \tag{4.6}$$

$$Error(X,Y,\theta,\vec{x},\vec{a})=\|\vec{a}^{\mathsf{T}}\cdot(\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}})\|^2=\frac{\|\vec{a}^{\mathsf{T}}\cdot(Mat(X,Y,t)\vec{x}^{\mathsf{T}})\|^2}{(1+t^2)^2}$$

$$Error(X,Y,\theta)=Error(X,Y,t)=\sum_i Error(X,Y,\theta,\vec{x}_i,\vec{a}_i)=$$

$$\sum_i\|\vec{a_i}^{\mathsf{T}}\cdot(\mathcal{T}(X,Y,\theta)\vec{x_i}^{\mathsf{T}})\|^2=\frac{\sum_i\|\vec{a_i}^{\mathsf{T}}\cdot(Mat(X,Y,t)\vec{x_i}^{\mathsf{T}})\|^2}{(1+t^2)^2} \tag{4.7}$$

For example, consider the case of the four points and corresponding three lines depicted in Figure 4.3. One pose which exactly maps the four points onto their corresponding features is given in Figure 4.4. The four points are: $(\frac{7}{2},\frac{-1}{2}),(\frac{9}{2},\frac{-1}{2}),(4+\frac{\sqrt{3}}{2},\frac{1}{2}),(4\Leftrightarrow\frac{\sqrt{3}}{2},\frac{1}{2})$, and the three associated lines are $x+1=0,\frac{1}{2}x\Leftrightarrow\frac{\sqrt{3}}{2}y\Leftrightarrow 1=0,\frac{1}{2}x+\frac{\sqrt{3}}{2}y\Leftrightarrow 1=0$; the first two points both correspond to the first line.

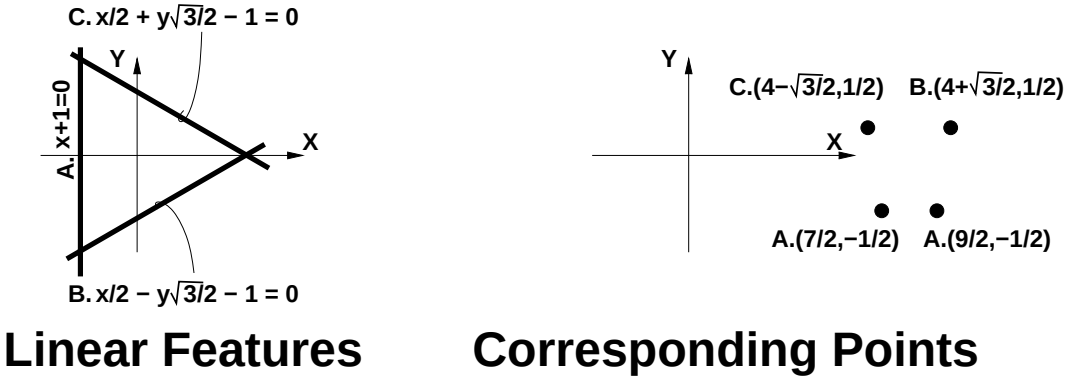


Figure 4.3: Four points corresponding to three linear features

In order to illustrate the concept of an error function, consider the pose $\theta=0,y=0$ (refer Figure 4.5). In this case, the error between the first point and first corresponding linear feature is $\frac{9}{2}$, and the error of the second point is $\frac{11}{2}$, and 1 for the both the third and fourth points.

To further illustrate the concept of the error function, in Figure 4.6, we depict the squared error function between the transformed points and the associated linear features as functions of y and t. The total error function $Error(Y,\theta)$, which is the sum of the four individual squared error functions is given in Figure 4.7. The task, which we use resultant

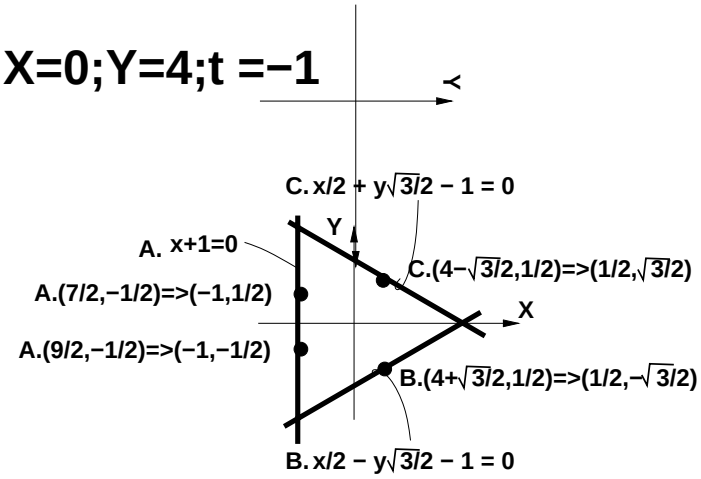


Figure 4.4: One solution transformation which maps the four points onto the corresponding linear features.

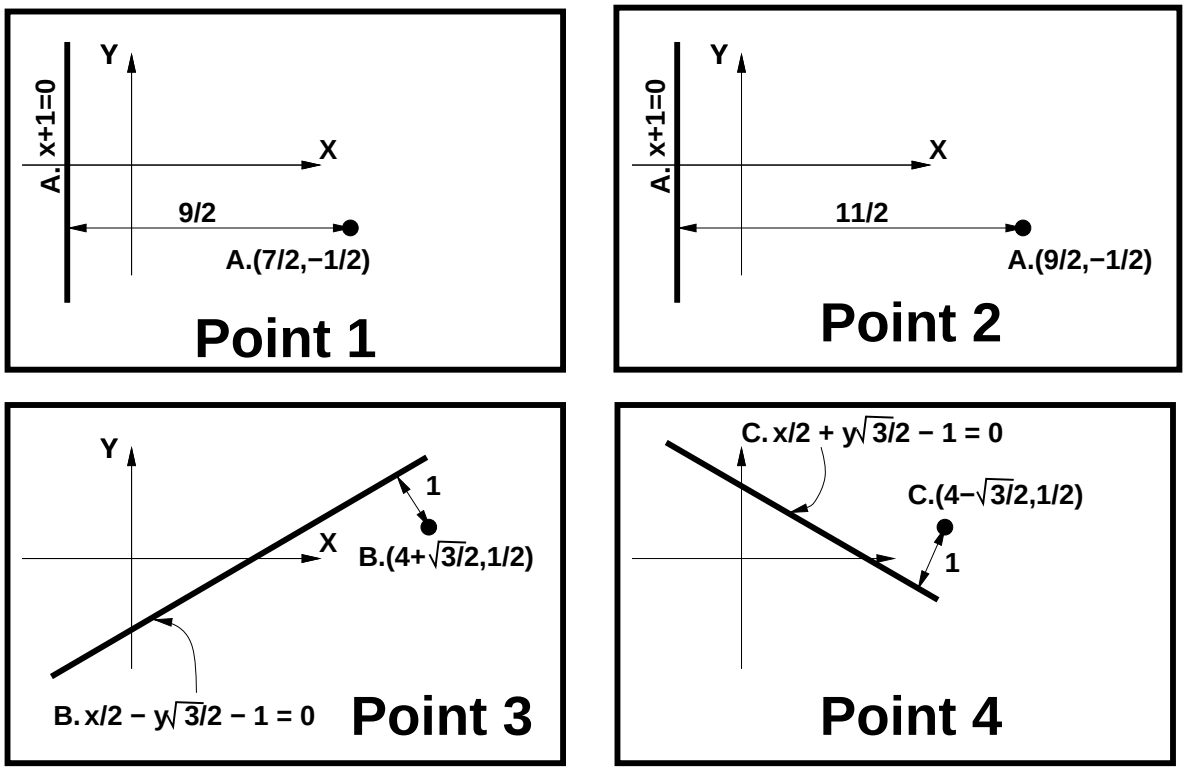
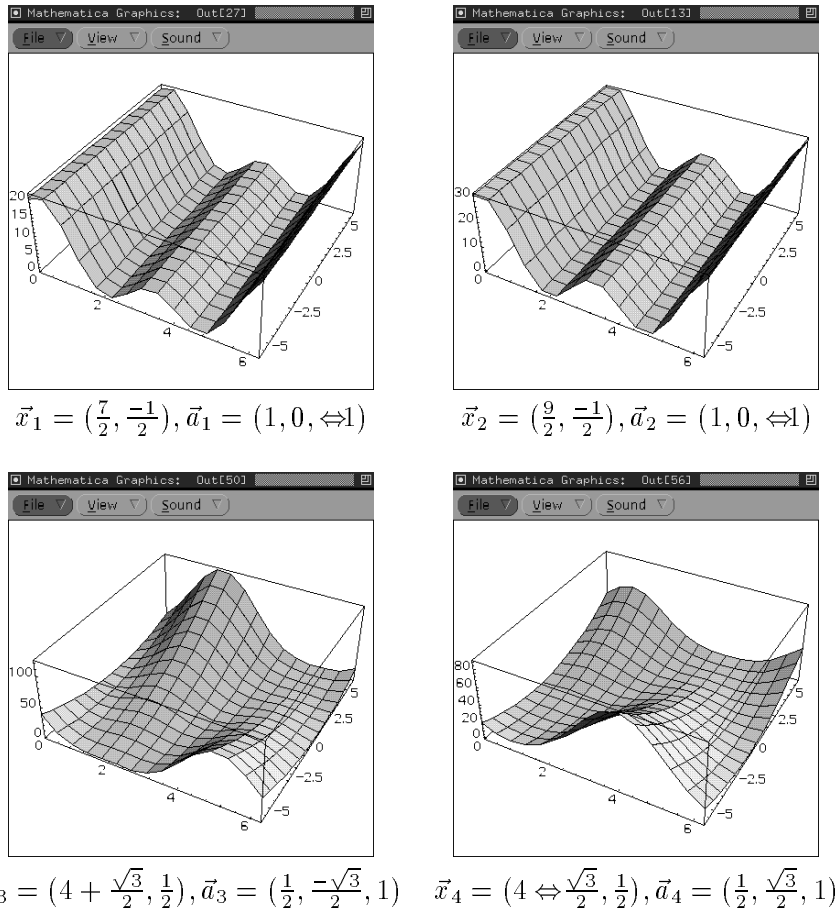


Figure 4.5: The distance between the points and the lines following the identity transformation $X = Y = t = 0$.

techniques to accomplish, is to efficiently find the global minimum of this error function; notice that the global minimum of the error function is zero at $\theta = \frac{3\pi}{2}$, $Y = 4$, which corresponds to mapping the vertices directly onto the corresponding linear features.



$\vec{x}_1 = (\frac{7}{2}, \frac{-1}{2}), \vec{a}_1 = (1, 0, \Leftrightarrow 1)$ $\vec{x}_2 = (\frac{9}{2}, \frac{-1}{2}), \vec{a}_2 = (1, 0, \Leftrightarrow 1)$

$\vec{x}_3 = (4 + \frac{\sqrt{3}}{2}, \frac{1}{2}), \vec{a}_3 = (\frac{1}{2}, \frac{-\sqrt{3}}{2}, 1)$ $\vec{x}_4 = (4 \Leftrightarrow \frac{\sqrt{3}}{2}, \frac{1}{2}), \vec{a}_4 = (\frac{1}{2}, \frac{\sqrt{3}}{2}, 1)$

Figure 4.6: The three dimensional plot of the error function $Error(Y, \theta, \vec{x}_i, \vec{a}_i)$ for each of the four points and corresponding linear features.

The sum of squared error between a set of points and corresponding lines is an algebraic function of the translation and rotation matrix $Mat(X, Y, t)$ (multiplied by $(1 + t^2)^2$). The coefficients $\mathtt{a}_{\mathtt{i}}\mathtt{y}_{\mathtt{j}}\mathtt{t}_{\mathtt{k}}$ are computable using the data set of points and associated linear features. The total sum squared error function (equation (4.7)) is multiplied by $(1 + t^2)^2$ to arrive at a polynomial $FF(X, Y, t)$ in X, Y and t (equations (4.8),(4.9)).

For each point $\vec{x}$ and associated linear feature $\vec{a}$, the function $FF(X, Y, t, \vec{x}, \vec{a})$ is a polynomial expression in X, Y and t. The total function $FF(X, Y, t)$ is the sum of all the individual $FF(X, Y, t, \vec{x}, \vec{a})$ error functions, and we can keep a running tally of all of the the

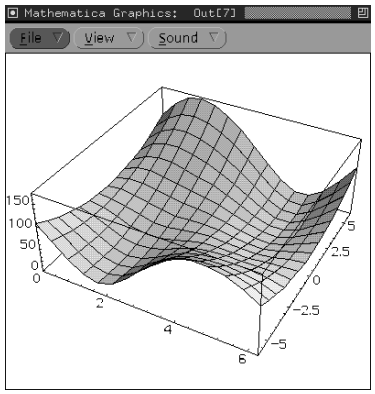


Figure 4.7: The total error function $Error(Y, \theta)$. The task, which we use resultant techniques to accomplish, is to efficiently find the global minimum of this error function.

symbolic coefficients (`a, ax1, ax1y1, ...`).

$$Error(X, Y, t) = \frac{FF(X, Y, t)}{(1 + t^2)^2} \tag{4.8}$$

$$\begin{aligned} FF(X, Y, t) = \sum_i \|\vec{a_i}^{\mathsf{T}} \cdot (Mat(X, Y, t)\vec{x_i}^{\mathsf{T}})\|^2 = {} & \mathtt{a} + X\mathtt{ax1} + XY\mathtt{ax1y1} + X^2\mathtt{ax2} + Y\mathtt{ay1} + \\ & Y^2\mathtt{ay2} + t\mathtt{at1} + Xt\mathtt{ax1t1} + Yt\mathtt{ay1t1} + t^2\mathtt{at2} + Xt^2\mathtt{ax1t2} + XYt^2\mathtt{ax1y1t2} + \\ & X^2t^2\mathtt{ax2t2} + Yt^2\mathtt{ay1t2} + Y^2t^2\mathtt{ay2t2} + t^3\mathtt{at3} + Xt^3\mathtt{ax1t3} + Yt^3\mathtt{ay1t3} + \\ & t^4\mathtt{at4} + Xt^4\mathtt{ax1t4} + XYt^4\mathtt{ax1y1t4} + X^2t^4\mathtt{ax2t4} + Yt^4\mathtt{ay1t4} + Y^2t^4\mathtt{ay2t4} \end{aligned} \tag{4.9}$$

In the case of polygonal objects, some of the algebraic coefficients are redundant because the generic algebraic expression includes coefficients which are dependent. We note these redundancies because they can be exploited in the resultant construction.

$$\begin{array}{cccc} \mathtt{ax1y1t2} = 2\,\mathtt{ax1y1} & \mathtt{ay2t2} = 2\,\mathtt{ay2} & \mathtt{ax2t2} = 2\,\mathtt{ax2} & \mathtt{ax2t4} = \mathtt{ax2} \\ \mathtt{ax1y1t4} = \mathtt{ax1y1} & \mathtt{ay2t4} = \mathtt{ay2} & \mathtt{ax1t3} = \mathtt{ax1t1} & \mathtt{ay1t3} = \mathtt{ay1t1} \end{array}$$

All of the minima of $Error(X, Y, t)$, both local and global, have partial derivatives (with respect to X, Y, t) equal to zero. To solve for the global minimum of $Error(X, Y, t)$, we want to find the configurations X, Y, t for which the partial derivatives are zero (refer equation (4.10)).

$$\nabla \cdot Error(X, Y, \theta) = (0, \vec{0}, 0) \Rightarrow \nabla \cdot Error(X, Y, t) = (0, \vec{0}, 0) \tag{4.10}$$

Resultant techniques require algebraic (not rational) functions. Therefore, we replace the *rational* partial derivatives with *equivalent algebraic* counterparts. Since we are only finding

the common roots of equation (4.10), we are at liberty to use any function $G_X(X,Y,\theta)$ for $\frac{\partial Error(X,Y,t)}{\partial X}$ where $\frac{\partial Error(X,Y,t)}{\partial X} = 0 \Rightarrow G_X(X,Y,\theta) = 0$ (and G_y for $\frac{\partial Error(X,Y,t)}{\partial Y}$ and G_t for $\frac{\partial Error(X,Y,t)}{\partial t}$) .

For $\frac{\partial Error(X,Y,t)}{\partial X}$ and $\frac{\partial Error(X,Y,t)}{\partial Y}$, $G_X(X,Y,t)$ and $G_Y(X,Y,t)$ are equal to the partial derivatives $\frac{\partial FF(X,Y,t)}{\partial X}$, $\frac{\partial FF(X,Y,t)}{\partial Y}$. $G_t(X,Y,t)$ is slightly more complicated because the denominator of $\frac{\partial Error(X,Y,t)}{\partial t}$ is a function of t, and that must be taken into account.

$$\begin{aligned}
\frac{\partial Error(X,Y,t)}{\partial X} &\propto \frac{\partial FF(X,Y,t)}{\partial X} \\
\Rightarrow G_X(X,Y,t) &= \frac{\partial FF(X,Y,t)}{\partial X} \\
&= 2X\texttt{ax2} + 4X\texttt{ax2}t^2 + 2X\texttt{ax2}t^4 \qquad (4.11)\\
&\quad + Y\texttt{ax1y1} + 2Y\texttt{ax1y1}t^2 + Y\texttt{ax1y1}t^4 \\
&\quad + \texttt{ax1} + \texttt{ax1t1}t + \texttt{ax1t2}t^2 + \texttt{ax1t3}t^3 + \texttt{ax1t4}t^4 \\
\frac{\partial Error(X,Y,t)}{\partial Y} &\propto \frac{\partial FF(X,Y,t)}{\partial Y} \\
\Rightarrow G_Y(X,Y,t) &= \frac{\partial FF(X,Y,t)}{\partial Y} \\
&= X\texttt{ax1y1} + 2X\texttt{ax1y1}t^2 + X\texttt{ax1y1}t^4 \\
&\quad 2Y\texttt{ay2} + 4Y\texttt{ay2}t^2 + 2Y\texttt{ay2}t^4 \\
&\quad \texttt{ay1} + \texttt{ay1t1}t + \texttt{ay1t2}t^2 + \texttt{ay1t3}t^3 + +\texttt{ay1t4}t^4 \qquad (4.12)\\
\frac{\partial Error(X,Y,t)}{\partial t} &\propto \frac{\partial \frac{FF(X,Y,t)}{(1+t^2)^2}}{\partial t} = \frac{((1+t^2)\frac{\partial FF(X,Y,t)}{\partial t} \Leftrightarrow 4tFF(X,Y,t))}{(1+t^2)^3} \qquad (4.13)\\
\Rightarrow G_t(X,Y,t) &= ((1+t^2)\frac{\partial FF(X,Y,t)}{\partial t} \Leftrightarrow 4tFF(X,Y,t)) \qquad (4.14)\\
\frac{\partial FF(X,Y,t)}{\partial t} &= \texttt{at1} + X\texttt{ax1t1} + Y\texttt{ay1t1} + 2\texttt{at2}t + 2X\texttt{ax1t2}t + 4XY\texttt{ax1y1}t \qquad (4.15)\\
&\quad + 4X^2\texttt{ax2}t + 2Y\texttt{ay1t2}t + 4Y^2\texttt{ay2}t + 3\texttt{at3}t^2 + 3X\texttt{ax1t3}t^2 + 3Y\texttt{ay1t3}t^2 \\
&\quad + 4\texttt{at4}t^3 + 4X\texttt{ax1t4}t^3 + 4XY\texttt{ax1y1}t^3 + 4X^2\texttt{ax2}t^3 \\
&\quad + 4Y\texttt{ay1t4}t^3 + 4Y^2\texttt{ay2}t^3 \\
G_t(X,Y,t) &= \texttt{ax1t1}X \Leftrightarrow 4\texttt{ax1}tX + 2\texttt{ax1t2}tX \Leftrightarrow 3\texttt{ax1t1}t^2X + 3\texttt{ax1t3}t^2X \Leftrightarrow 2\texttt{ax1t2}t^3X \\
&\quad + 4\texttt{ax1t4}t^3X \Leftrightarrow \texttt{ax1t3}t^4X \\
&\quad + \texttt{ay1t1}Y \Leftrightarrow 4\texttt{ay1}tY + 2\texttt{ay1t2}tY \Leftrightarrow 3\texttt{ay1t1}t^2Y + 3\texttt{ay1t3}t^2Y \Leftrightarrow 2\texttt{ay1t2}t^3Y \\
&\quad + 4\texttt{ay1t4}t^3Y \Leftrightarrow \texttt{ay1t3}t^4Y \\
&\quad + \texttt{at1} \Leftrightarrow 4\texttt{a}t + 2\texttt{at2}t \Leftrightarrow 3\texttt{at1}t^2 + 3\texttt{at3}t^2 \Leftrightarrow 2\texttt{at2}t^3 + 4\texttt{at4}t^3 \Leftrightarrow \texttt{at3}t^4 \qquad (4.16)
\end{aligned}$$

Example and Explanation of Resultant Methods

Elimination of X, Y from this system of equations (equation (4.11) = 0, equation (4.12) = 0, equation (4.16) = 0) expresses the resultant as the determinant of a matrix polynomial in t. We now show how these systems are solved using elimination theory. Consider the three functions $G_X(X,Y,t)$, $G_Y(X,Y,t)$, $G_t(X,Y,t)$ which are proportional to the partial derivatives: $\frac{\partial Error(X,Y,t)}{\partial X}$, $\frac{\partial Error(X,Y,t)}{\partial Y}$, $\frac{\partial Error(X,Y,t)}{\partial t}$. Notice that $G_X(X,Y,t)$, $G_Y(X,Y,t)$, $G_t(X,Y,t)$ are all quartic in t, *but linear in* X *and* Y. Resultants can be constructed for any algebraic system of equations, but they are simplest for cases in which the equations are linear in all but a single variable. We can rewrite the three equations as linear functions of X,Y, and t:

$$\begin{aligned} G_X(X,Y,t) &= f_{11}(t)X + f_{12}(t)Y + f_{13}(t) = 0 \\ G_Y(X,Y,t) &= f_{21}(t)X + f_{22}(t)Y + f_{23}(t) = 0 \\ G_t(X,Y,t) &= f_{31}(t)X + f_{32}(t)Y + f_{33}(t) = 0 \end{aligned}$$

Consider the matrix equation (equation 4.17); in order for the partial derivatives to be simultaneously zero, $(X,Y,1)^{\mathsf{T}}$ must be in the null space of the *resultant* matrix (refer equation (4.17)). The resultant matrix has a null space if and only if the determinant of the resultant matrix is zero. Furthermore, any solution to the system of equations must lie in the null space of the resultant matrix. We have therefore derived a relationship between roots of the system of equations, and a single expression in a single variable. If t is a coordinate of a simultaneous solution of this system of equations, then the determinant of the resultant matrix as a function of t must be zero. We can solve for all values of t using the resultant matrix, by finding all t such that $|ResultantMatrix(t)| = 0$ (where $|ResultantMatrix(t)|$ signifies the determinant of $ResultantMatrix(t)$) and then we can solve for X_t and Y_t for each t.

$$\begin{bmatrix} G_X(X,Y,t) \\ G_Y(X,Y,t) \\ G_t(X,Y,t) \end{bmatrix} = \underbrace{\begin{bmatrix} f_{11}(t) & f_{12}(t) & f_{13}(t) \\ f_{21}(t) & f_{22}(t) & f_{23}(t) \\ f_{31}(t) & f_{32}(t) & f_{33}(t) \end{bmatrix}}_{ResultantMatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{4.17}$$

Solving the system of equations is reduced to finding roots of the determinant of a matrix polynomial, where $M(t)$ denotes the resultant matrix.

$$M(t) = Resultant(\{\frac{\partial Error(X,Y,t)}{\partial X}, \frac{\partial Error(X,Y,t)}{\partial Y}, \frac{\partial Error(X,Y,t)}{\partial t}\}, \{X,Y\})$$

$$= Resultant(\{G_X(X,Y,t), G_Y(X,Y,t), G_t(X,Y,t)\}, \{X,Y\}) \quad (4.18)$$

$$= \begin{bmatrix} 2\mathtt{ax2} + 4\mathtt{ax2}t^2 & \mathtt{ax1y1} + 2\mathtt{ax1y1}t^2 & \mathtt{ax1} + \mathtt{ax1t1}t + \mathtt{ax1t2}t^2 \\ +2\mathtt{ax2}t^4 & +\mathtt{ax1y1}t^4 & \mathtt{ax1t3}t^3 + \mathtt{ax1t4}t^4 \\ \\ \mathtt{ax1y1} + 2\mathtt{ax1y1}t^2 & 2\mathtt{ay2} + 4\mathtt{ay2}t^2 & \mathtt{ay1} + \mathtt{ay1t1}t + \mathtt{ay1t2}t^2 \\ +\mathtt{ax1y1}t^4 & +2\mathtt{ay2}t^4 & \mathtt{ay1t3}t^3 + \mathtt{ay1t4}t^4 \\ \\ \mathtt{ax1t1} \Leftrightarrow 4\mathtt{ax1t1}t & \mathtt{ay1t1} \Leftrightarrow 4\mathtt{ay1t1}t & \mathtt{at1} \Leftrightarrow 4\mathtt{a}t + 2\mathtt{at2}t \\ +2\mathtt{ax1t2}t \Leftrightarrow 3\mathtt{ax1t1}t^2 & +2\mathtt{ay1t2}t \Leftrightarrow 3\mathtt{ay1t1}t^2 & \Leftrightarrow 3\mathtt{at1}t^2 + 3\mathtt{at3}t^2 \Leftrightarrow 2\mathtt{at2}t^3 \\ +3\mathtt{ax1t3}t^2 \Leftrightarrow 2\mathtt{ax1t2}t^3 & +3\mathtt{ay1t3}t^2 \Leftrightarrow 2\mathtt{ay1t2}t^3 & +4\mathtt{at4}t^3 \Leftrightarrow \mathtt{at3}t^4 \\ +4\mathtt{ax1t4}t^3 \Leftrightarrow \mathtt{ax1t3}t^4 & +4\mathtt{ay1t4}t^3 \Leftrightarrow \mathtt{ay1t3}t^4 & \end{bmatrix}$$

4.4.3 Solving the Matrix Polynomial

$M(t)$ was obtained (equation (4.18)) by eliminating X and Y from the equations corresponding to the partial derivatives of $Error(X,Y,t)$. In this section, we highlight a numerically accurate and efficient algorithm to compute the roots of $|M(t)| = 0$. All of the t coordinates of the common solutions of the algebraic equations are roots of $|M(t)| = 0$.

A simple procedure for root finding involves expanding the symbolic determinant and computing the roots of the resulting univariate polynomial. This procedure involves a significant amount of symbolic computation (in determinant expansion), which makes it relatively slow. Moreover, the problem of computing roots of polynomials of degree greater than 10 are can be numerically ill-conditioned. As a result, the solutions obtained using IEEE floating point arithmetic (or any other model of finite precision arithmetic) are likely to be numerically inaccurate. To circumvent these problems, we use matrix computations.

Solving $|M(t)| = 0$ is reduced to determining the eigenvalues of the block companion matrix E. The construction of the block companion matrix E is highlighted in equation (4.19). It involves matrix inversion and matrix products. In this section we assume that the leading matrix, M_4, is numerically well-conditioned. The eigenvalues of E are exactly the roots of the equations $|M(t)| = 0$ [Man92] computed using routines from LAPACK [ABB+92] and EISPACK [GBDM77]. Solving for the roots of $M(t)$ in this manner is numerically better conditioned than solving the resultant polynomial formed by expanding

the resultant matrix.

$$M(t) = M_4t^4 + M_3t^3 + M_2t^2 + M_1t + M_0 \tag{4.19}$$

$$E = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \\ M_4^{-1}M_0 & M_4^{-1}M_1 & M_4^{-1}M_2 & M_4^{-1}M_3 \end{bmatrix} \tag{4.20}$$

Identifying Possibly Valid Solutions

Resultant formulations can generate extraneous solutions; i.e., roots of the resultant which have no corresponding solutions in the original system of equations. The normal approach is to consider each root of the resultant a candidate solution. The disadvantage of treating all solutions as candidates is that the resultant formalism generates complex solutions even when only real solutions are desired. In this case of matching points to linear features, the orientation t is expected to be real. Furthermore, the resultant construction introduces multiple extraneous roots $i, i, \Leftrightarrow i, \Leftrightarrow i$, and this is seen by the fact that the symbolic resultant's characteristic polynomial is divisible by $(1+t^2)^2$. We ignore complex solutions by thresholding the imaginary components.

Improving the Accuracy

The eigenvalue decomposition depends upon the invertibility of the largest exponent matrix (in this case, M_4 is well conditioned and the inverse computation does not lead to ill-conditioned matrices). Sometimes, the highest degree matrix M_4 is symbolically rank deficient. In this case, we use an algebraic substitution ($s = f(t)$), and generate a matrix polynomial M' such that $M'(s) = 0 \Leftrightarrow M(t) = 0$ with an invertible largest exponent matrix M_4' ($M_4' = Coefficient[M', s^4]$) (where $Coefficient[\mathcal{E}, t^i]$ describes the coefficient of the t^i term in the expression $\mathcal{E}$). Two simple transformations are the geometric inverse ($t = \frac{1}{s}$) and a random generic rational transformation ($t = \frac{\alpha s+\beta}{\gamma s+\delta}$). We first try the geometric inverse substitution $t = \frac{1}{s}$ which corresponds to checking whether M_0 is numerically well-conditioned.

Let $\mathbf{d}_\mathrm{M}$ be M's polynomial degree. If $M_0(t)$ is not well conditioned, we instead try a number of generic rational transformations: $t = \frac{\alpha s+\beta}{\gamma s+\delta}$, with random $\alpha, \beta, \gamma, \delta$. If any of these transformations produces a well conditioned matrix M_4' signified by its condition

number, we use the random transformation corresponding to the best conditioned matrix. We compute the eigenvalues s of the related large matrix E' (the companion matrix of the M' matrix), to find all of the s for which $|M'(s)| = 0$. The t values for which $|M(t)| = 0$ are computed by applying the inverse function f^{-1} to the s values.

$$\begin{aligned} t = \frac{1}{s} &\rightarrow M_{t=1/s}(s) = s^{\mathbf{d}_M} M(t) \\ &\rightarrow Coefficient[M_{t=1/s}, s^d] = Coefficient[M, t^{\mathbf{d}_M - d}] \\ t = \frac{\alpha s + \beta}{\gamma s + \delta} &\rightarrow M_{t=\frac{\alpha s+\beta}{\gamma s+\delta}}(s) = M(t)(\gamma s + \delta)^{\mathbf{d}_M} \end{aligned} \tag{4.21}$$

If none of the rational transformations lead to numerically invertible largest exponent matrices, we can use the generalized eigenvalue formulation.

Generalized Eigenvalue Formulation

In case M_4 is singular or numerically ill-conditioned and none of the transformations produces a well conditioned matrix, we reduce root finding to a generalized eigenvalue problem of a matrix pencil [GL89]. The matrix pencil for the generalized eigenvalue problem is constructed from the matrix polynomial in the following manner. Let

$$C_1 = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & M_4 \end{bmatrix} \quad C_2 = \begin{bmatrix} 0 & \Leftrightarrow I & 0 & 0 \\ 0 & 0 & \Leftrightarrow I & 0 \\ 0 & 0 & 0 & \Leftrightarrow I \\ M_0 & M_1 & M_2 & M_3 \end{bmatrix} \tag{4.22}$$

and the eigenvalues of $C_1 t + C_2$ correspond exactly to the roots of $|M(t)| = 0$.

4.4.4 Computing X_t and Y_t

Given the value of t computed using eigenvalue routines, the corresponding X_t and Y_t coordinates of the roots of the algebraic equations are found in constant time by solving $\frac{\partial Error(X,Y,t)}{\partial X}|_t = 0$ and $\frac{\partial Error(X,Y,t)}{\partial Y}|_t = 0$. The two linear equations in X and Y can be solved using Cramer's rule.

Verifying That (X_t, Y_t, t) is a Local Extremum

The method highlighted above is necessary *but not sufficient* and can produce extraneous solutions. Therefore, we need to verify that each (X_t, Y_t, t) configuration is

actually a local extremum. This is accomplished by computing the absolute values of the partial derivatives of the error function and comparing them to some ϵ value, where the ϵ should be non-zero to accommodate sensor noise and numerical error.

4.4.5 Example

In this section, we step through an example in order to illustrate how we use resultants to solve the non-linear least squares problem. Notice that we do not recompute $G_x(X,Y,t)$, $G_Y(X,Y,t)$, or $G_t(X,Y,t)$ online. $G_x(X,Y,t)$, $G_Y(X,Y,t)$, or $G_t(X,Y,t)$ were intermediate offline results used to construct the symbolic resultant matrix. In other words, the symbolic resultant matrix is the *compiled* result of the elimination. For brevity, we present the intermediary numeric values to four decimal places, though we use double precision arithmetic.

The resultant computation is comprised of seven steps:

1. Compute the symbolic coefficients using the data set
2. Construct the symbolic resultant using the symbolic coefficients
3. If necessary, apply transformations in order to find a well conditioned invertible highest exponent matrix
4. Construct the companion matrix using the resultant matrix
5. Compute the eigenvalues of the companion matrix
6. If the algorithm applied a transformation in step 3, find the roots of t by applying an inverse function to the eigenvalues
7. Compute the remaining pose parameters, X_t,Y_t for each candidate, t

Evaluating the Symbolic Coefficients

The data set of points and associated linear features is given in Table 4.1. The symbolic coefficients' values shown in Table 4.2 are computed from the points and features. They describe the coefficients of the sum of the algebraic error functions (refer equation(4.23)).

Data point $(x_{i,1}, x_{i,2})$	Linear Feature Algebraic Parameters $(a_{i,1}, a_{i,2}, a_{i,3})$
(-7.91, -7.91)	(-0.007534555543, 0.999971614834, -9.004401406730)
(7.91, 7.91)	(-0.007534555543, 0.999971614834, 6.805099825207)
(-7.91, 7.91)	(0.700109199157, 0.714035789899, -12.166817390266)
(7.91, -7.91)	(0.700109199157, 0.714035789899, 10.050656124962)
(-7.91, -7.91)	(-0.710861891474, 0.703331622529, -11.545580060073)
(7.91, 7.91)	(-0.710861891474, 0.703331622529, 10.561258166615)

Table 4.1: Example data set: data points and corresponding algebraically parameterized linear features. The task is to compute the transformation which optimally transforms the set of points onto the corresponding linear features.

$\mathtt{a}$	503.8720	$\mathtt{at}_4$	985.2446	$\mathtt{ax}_1\mathtt{t}_4$	1.5305	$\mathtt{ay}_1\mathtt{t}_1$	0
$\mathtt{at}_1$	-2001.9285	$\mathtt{ax}_1$	1.5305	$\mathtt{ax}_2$	1.9911	$\mathtt{ay}_1\mathtt{t}_2$	17.6102
$\mathtt{at}_2$	3506.1299	$\mathtt{ax}_1\mathtt{t}_1$	0	$\mathtt{ax}_1\mathtt{y}_1$	-0.0304	$\mathtt{ay}_1\mathtt{t}_4$	8.8051
$\mathtt{at}_3$	-2972.5279	$\mathtt{ax}_1\mathtt{t}_2$	3.0610	$\mathtt{ay}_1$	8.8051	$\mathtt{ay}_2$	4.0089

Table 4.2: The values of the symbolic coefficients which are used in the resultant formulation: $FF(X,Y,t) = \mathtt{a} + \mathtt{at1}t + \ldots$

We present $FF(X,Y,t,(\Leftrightarrow 7.91, \Leftrightarrow 7.91, 0.0),(0.0075, 1.000, \Leftrightarrow 9.0044))$ in equation (4.24) and Figure 4.8.

$$\begin{aligned} FF(X,Y,t) \;=\; & FF(X,Y,t,(\Leftrightarrow 7.91, \Leftrightarrow 7.91, 0.0),(0.0075, 1.000, \Leftrightarrow 9.0044)) \\ & + FF(X,Y,t,(7.91, 7.91, 0.0),(0.0075, 1.000, 6.8041)) \\ & + FF(X,Y,t,(\Leftrightarrow 7.91, 7.91, 0.0),(0.7001, 0.7140, \Leftrightarrow 12.1668)) + \ldots \end{aligned} \tag{4.23}$$

$$\begin{aligned} & FF(X,Y,t,(\Leftrightarrow 7.91, \Leftrightarrow 7.91, 0.0),(0.0075, 1.000, \Leftrightarrow 9.0044)) = \\ & \quad 1.3322 \Leftrightarrow 36.7938t + 292.9516t^2 \Leftrightarrow 537.2817t^3 + 284.0768t^4 \\ & \quad + 2.3084Y \Leftrightarrow 31.8766Yt + 36.0166Yt^2 \Leftrightarrow 31.8766Yt^3 + 33.7082Yt^4 \\ & \quad + 0.9999(1+t^2)^2Y^2 + 0.0001(1+t^2)^2X^2 \Leftrightarrow 0.0151(1+t^2)^2XY \\ & \quad \Leftrightarrow 0.0174X + 0.2402Xt \Leftrightarrow 0.2714Xt^2 + 0.2402Xt^3 \Leftrightarrow 0.2540Xt^4 \end{aligned} \tag{4.24}$$

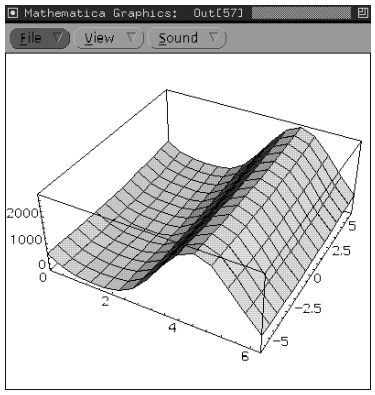


Figure 4.8: The total error function $Error(Y, \theta)$ for this example

Constructing the Matrix Resultant M Using the Symbolic Coefficients

The resultant $M(t)$ in equation (4.25) is generated from equation (4.18).

$$M(t) = \begin{bmatrix} 3.9821\mathrm{t}^4 + 7.9643\mathrm{t}^2 + 3.9821 & \Leftrightarrow 0.0304t^4 \Leftrightarrow 0.0608t^2 \Leftrightarrow 0.0304, & 0 \\ \Leftrightarrow 0.0304t^4 \Leftrightarrow 0.0608t^2 \Leftrightarrow 0.0304 & 8.0179t^4 + 16.0357t^2 + 8.0179 & 0 \\ 1.5305t^4 + 3.0610t^2 + 1.5305 & 8.8051t^4 + 17.6102t^2 + 8.8051 & \psi \end{bmatrix} \quad (4.25)$$

$$\psi = 2972.5279t^4 \Leftrightarrow 3071.2816t^3 \Leftrightarrow 2911.7982t^2 + 4996.7719t \Leftrightarrow 2001.9285$$

Constructing the Companion Matrix E

At this point, we need to determine whether the highest exponent matrix M_4 is numerically invertible. We check its invertibility by computing the condition number for a particular random value of t (in this case, 0.396465). $M_4(t = 0.396465)$ (equation (4.26)) is obviously not uniformly degenerate since the matrix is well conditioned for a random value of t.

$$M_4(t = 0.396465) = \begin{bmatrix} 5.3324 & \Leftrightarrow 0.0407 & 0 \\ \Leftrightarrow 0.0407 & 10.7365 & 0 \\ 2.0495 & 11.7907 & \Leftrightarrow 596.5278 \end{bmatrix} \quad (4.26)$$

Computing the companion matrix E of M involves inverting the highest degree matrix (in this case M_4) and then left-multiplying all of the matrixes by M_4^{-1}. We compute

$M_4^{-1}M_0 \ldots M_4^{-1}M_3$ and combine them with an identity matrix to construct E.

$$M_4^{-1} = \begin{bmatrix} 0.2511 & 0.0010 & 0 \\ 0.0010 & 0.1247 & 0 \\ \Leftrightarrow 0.0001 & \Leftrightarrow 0.0004 & 0.0003 \end{bmatrix}; M_4^{-1}M_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \Leftrightarrow 0.6735 \end{bmatrix}$$

$$M_4^{-1}M_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1.6810 \end{bmatrix}; M_4^{-1}M_2 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & \Leftrightarrow 0.9796 \end{bmatrix}; M_4^{-1}M_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Leftrightarrow 1.0332 \end{bmatrix}$$

$$E = \begin{bmatrix} \mathbf{0}^{3\times 9} & & & & & & & & & \mathbf{I}^{9\times 9} & & \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \Leftrightarrow 1.0332 & 0 & 0 & \Leftrightarrow 0.9796 & 0 & 0 & 1.6810 & 0 & 0 & \Leftrightarrow 0.6735 \end{bmatrix}$$

Determining Roots of the Matrix Polynomial by Computing the Eigenvalues of Companion Matrix E

The eigenvalues of E are given in equation (4.27). Some of the 12 eigenvalues of E (E is a square 12×12 matrix) are complex (with imaginary components $> \epsilon$). This is due to the fact that the algebraic formulation of the transformation $Mat(X, Y, t)$ defines a function for complex values of t as well.

$$Eigenvalues(E) = \begin{Bmatrix} 0.000000009726 + 1.000000009241\,\mathrm{i} & 0.000000009726 + -1.000000009241\,\mathrm{i} \\ -0.000000009726 + 0.999999990759\,\mathrm{i} & -0.000000009726 + -0.999999990759\,\mathrm{i} \\ -1.231429745741 + 0.000000000000\,\mathrm{i} & -0.000000005427 + 1.000000002712\,\mathrm{i} \\ -0.000000005427 + -1.000000002712\,\mathrm{i} & 0.000000005427 + 0.999999997288\,\mathrm{i} \\ 0.000000005427 + -0.999999997288\,\mathrm{i} & 1.008279326594 + 0.000000000000\,\mathrm{i} \\ 0.628186277667 + 0.384444450582\,\mathrm{i} & 0.628186277667 + -0.384444450582\,\mathrm{i} \end{Bmatrix} \quad (4.27)$$

Computing X_t and Y_t for Candidate Solutions

We compute the remaining pose parameters (X_t, Y_t) for each candidate t, the real eigenvalues, $\{\Leftrightarrow 1.231429745741, 1.008279326594\}$. The values of X_t, Y_t, and $Error(X_t, Y_t, t)$ for $t = \Leftrightarrow 1.231429745741$ and $t = 1.008279326594$ are given in Table 4.3. The error function $Error(X_t, Y_t, t)$ at $t = 1.008279326594$ is extremely small, and we observe that it is a local minimum from the signs of the second order partial derivatives of $Error(X_t, Y_t, t)$. The

Type	t	X_t	Y_t	$Error(X_t, Y_t, t)$
Local Maximum	-1.231429745741	-0.392742825743	-1.099677271638	2537.708141328489
Local Minimum	1.008279326594	-0.392742825743	-1.099677271638	0.047461161151

Table 4.3: Local extrema found by solving $\nabla \cdot Error(X, Y, t) = 0$

Data point $(x_{i,1}, x_{i,2})$	Linear Feature Algebraic Parameters $(a_{i,1}, a_{i,2}, a_{i,3})$
(-7.91, -7.91)	(-0.007534555543, 0.999971614834, -9.004401406730)
(7.91, 7.91)	(-0.007534555543, 0.999971614834, 6.805099825207)
(-7.91, 7.91)	**(-0.007534555543, 0.999971614834, 6.805099825207)**
(7.91, -7.91)	(0.700109199157, 0.714035789899, 10.050656124962)
(-7.91, -7.91)	(-0.710861891474, 0.703331622529, -11.545580060073)
(7.91, 7.91)	(-0.710861891474, 0.703331622529, 10.561258166615)

Table 4.4: Incorrect data set: an incorrect correspondence is included to demonstrate the localization technique's robustness.

remaining candidates are disregarded because they are complex. Since the global minimum is guaranteed to be one of the local extrema, the pose corresponding to $t = 1.008279326594$ is in fact the global minimum.

The pose is completed by transforming the t value into radians using equation (4.28). In this case, $\theta(t = 1.008279326594) = 1.5790415$ radians.

$$\theta(t) = 2\tan^{-1}(t) \tag{4.28}$$

4.4.6 Effect of Incorrect Correspondence

In this section, we show that the algorithm computes an optimum pose estimate even when the correspondence information is incorrect. The algorithm computes the transformation which maps the point set onto the feature set, regardless of the quality of the fit between the transformed point set and the feature set. To demonstrate its robustness, we applied the localization algorithm on almost the same data set with one exception: the third point now corresponds to the same linear feature as the first two points (refer Table 4.4). A slice of the error function is depicted in Figure 4.9, and the poses with extremal squared error are given in Table 4.5.

Therefore, a high error residual at the local minima would indicate a wrong correspondence.

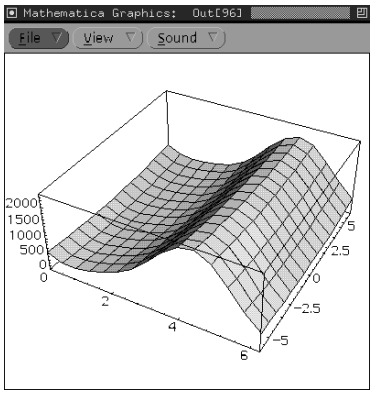


Figure 4.9: The total error function $Error(Y, \theta)$ given the wrong correspondence information. Notice that the minimum sum squared error is on the order of 82, inferring incorrect correspondence.

Type	t	X_t	Y_t	$Error(X_t, Y_t, t)$
Local Maximum	-1.318194808282	11.674225332740	4.323041075059	1778.848128685287
Local Minimum	0.391742823602	2.115870897468	1.402893370181	82.262413290594

Table 4.5: Extremal error poses found for the data set with the incorrect correspondences.

4.4.7 Example Which Demonstrates Problems of Local Minima

The main advantage of using an algebraic approach to solve the non-linear least squares problem is that it always returns the global minimum, not a local minimum. Gradient descent techniques can fail by returning a local minimum. Table 4.6 shows a set of points and linear feature parameters which induce a local minimum of the error function. The error function is depicted in Figure 4.10, and the local minima are given in Table (4.7). In the case of linear features, it is impossible for a single value of t to correspond to two local minima because $\frac{\partial Error(X,Y,t)}{\partial X}|_t = 0$ and $\frac{\partial Error(X,Y,t)}{\partial Y}|_t = 0$ are linear equations in X and Y (allowing only a single generic solution).

In the example given in section 4.4.5, notice that there are only four non-degenerate values of t (refer Table 4.7); in the example given in section 4.4.7, there are also only four non-degenerate values of t.

4.4.8 Points and Rectilinear Features

Having gone through the examples for localizing generic points and linear features, we now describe a localization algorithm optimized for points and rectilinear features. This

Model point $(x_{i,1}, x_{i,2})$	Algebraic Parameters $(a_{i,1}, a_{i,2}, a_{i,3})$
(1.0, 0.0)	(1.0, 0.0, 1.0)
(0.0, 1.0)	(0.0, 1.0, 1.0)
(-1.0, 0.0)	(1.0, 0.0, -1.0)
(0.0, -1.0)	(0.0, 1.0, -1.0)
(0.6, 0.87)	(3.06, 3.52, 4.09)

Table 4.6: An example of points and corresponding linear features which induce a local minima in the error function.

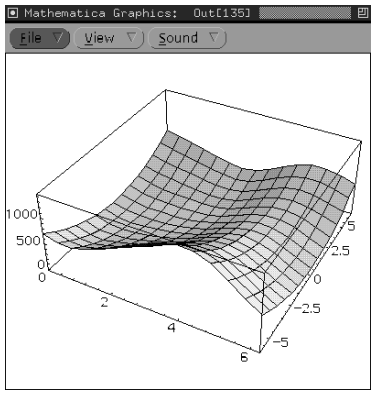


Figure 4.10: The total error function $Error(Y, \theta)$ for the example with local minima

Type	t	X_t	Y_t	$Error(X_t, Y_t, t)$
Local Minimum	0.163456292485	-0.048729446432	-0.056054788052	0.022882658439
Local Minimum	-0.159961402498	-0.094761199366	-0.109006346984	0.055519581104
Local Maxima	-56.167085506603	1.160012442320	1.33439339770	22.817164552496
Local Maxima	-0.061241409474	-0.108071632540	-0.124317694947	0.059481427413

Table 4.7: Local extrema found by solving $\nabla \cdot Error(X, Y, t) = 0$

tailored algorithm is much faster than the general case algorithm because the symbolic resultant can be simplified to a fourth order equation when all of the linear features are rectilinear, i.e. either parallel or normal to each other. By redesigning the parts, or concentrating on only the rectilinear edges, this special case can be exploited to achieve real-time localization with cheaper computer hardware.

Although parallel edges can be represented in any manner, the description for which they are all parallel either to the x axis or the y axis is particularly beneficial because some of the symbolic coefficients become redundant, and this redundancy allows us to simplify the symbolic resultant. For example the `ax1t2` coefficient (for the Xt^2 term in $FF(X,Y,t)$) is exactly the sum of the `ax1` coefficient (for the X term in $FF(X,Y,t)$) and the `ax1t4` coefficient (for the Xt^4 term in $FF(X,Y,t)$); although these redundancies are unintuitive, they are observed by expanding the error between points and horizontal lines (refer equation (4.6)), and recognizing them is beneficial. The redundant formulation of $FF(X,Y,t)$ is given in equation (4.29).

$$\texttt{ax1t2} = \texttt{ax1} + \texttt{ax1t4}; \quad \texttt{ay1t2} = \texttt{ay1} + \texttt{ay1t4}; \quad \texttt{ax1y1} = \texttt{ax1y1t2} = \texttt{ax1y1t4} = 0$$

$$\begin{aligned} FF(X,Y,t) \;=\; & \texttt{a} + \texttt{at1}t + \texttt{at2}t^2 + \texttt{at3}t^3 + \texttt{at4}t^4 + \texttt{ax1}X + \texttt{ax1t1}Xt \\ & + \texttt{ax1}Xt^2 + \texttt{ax1t4}Xt^2 + \texttt{ax1t1}Xt^3 + \texttt{ax1t4}Xt^4 + \texttt{ax2}X^2 \\ & + 2\texttt{ax2}X^2t^2 + \texttt{ax2}X^2t^4 + \texttt{ay1}Y + \texttt{ay1t1}Yt + \texttt{ay1}Yt^2 \\ & + \texttt{ay1t4}Yt^2 + \texttt{ay1t1}Yt^3 + \texttt{ay1t4}Yt^4 + \texttt{ay2}Y^2 + 2\texttt{ay2}Y^2t^2 + \texttt{ay2}Y^2t^4 \end{aligned} \tag{4.29}$$

In the case of rectilinear features, the resultant polynomial is a twelfth degree polynomial in t, but it can be factored into a fourth order polynomial and eighth order polynomial (with eight extraneous solutions, refer equation (4.30)). Thereby rectilinear features can be localized faster because the fourth order polynomial in t can be solved exactly in constant time [CRC73].

$$\begin{aligned} \frac{|M(t)|}{(1+t^2)^4} &= a_0 + a_1t + a_2t^2 + a_3t^3 + a_4t^4 \qquad (4.30) \\ a_0 &= \Leftrightarrow 2 * \texttt{ax2} * \texttt{ay1} * \texttt{ay1t1} \Leftrightarrow 2 * \texttt{ax1} * \texttt{ax1t1} * \texttt{ay2} + 4 * \texttt{at1} * \texttt{ax2} * \texttt{ay2} \\ a_1 &= 4 * \texttt{ax2} * \texttt{ay1}^2 \Leftrightarrow 2 * \texttt{ax2} * \texttt{ay1t1}^2 \Leftrightarrow 4 * \texttt{ax2} * \texttt{ay1} * \texttt{ay1t4} + 4 * \texttt{ax1}^2 * \texttt{ay2} \\ &\quad \Leftrightarrow 2 * \texttt{ax1t1}^2 * \texttt{ay2} \Leftrightarrow 4 * \texttt{ax1} * \texttt{ax1t4} * \texttt{ay2} \Leftrightarrow 16 * \texttt{a} * \texttt{ax2} * \texttt{ay2} \\ &\quad + 8 * \texttt{at2} * \texttt{ax2} * \texttt{ay2} \\ a_2 &= 6 * \texttt{ax2} * \texttt{ay1} * \texttt{ay1t1} \Leftrightarrow 6 * \texttt{ax2} * \texttt{ay1t1} * \texttt{ay1t4} + 6 * \texttt{ax1} * \texttt{ax1t1} * \texttt{ay2} \end{aligned}$$

$$
\begin{aligned}
& \Leftrightarrow 6 * \texttt{ax1t1} * \texttt{ax1t4} * \texttt{ay2} \Leftrightarrow 12 * \texttt{at1} * \texttt{ax2} * \texttt{ay2} + 12 * \texttt{at3} * \texttt{ax2} * \texttt{ay2} \\
a_3 \;=\; & 2 * \texttt{ax2} * \texttt{ay1t1}^2 + 4 * \texttt{ax2} * \texttt{ay1} * \texttt{ay1t4} \Leftrightarrow 4 * \texttt{ax2} * \texttt{ay1t4}^2 + 2 * \texttt{ax1t1}^2 * \texttt{ay2} \\
& + 4 * \texttt{ax1} * \texttt{ax1t4} * \texttt{ay2} \Leftrightarrow 4 * \texttt{ax1t4}^2 * \texttt{ay2} \Leftrightarrow 8 * \texttt{at2} * \texttt{ax2} * \texttt{ay2} \\
& + 16 * \texttt{at4} * \texttt{ax2} * \texttt{ay2} \\
a_4 \;=\; & 2 * \texttt{ax2} * \texttt{ay1t1} * \texttt{ay1t4} + 2 * \texttt{ax1t1} * \texttt{ax1t4} * \texttt{ay2} \Leftrightarrow 4 * \texttt{at3} * \texttt{ax2} * \texttt{ay2} \quad (4.31)
\end{aligned}
$$

4.5 Optimal Pose Determination for Models with Circular and Linear Features

In section 4.4 we described the localization algorithm for polygonal objects and objects with linear features. In this section, we extend the algorithm to the larger class of generalized-polygonal objects: objects with boundaries consisting of linear and circular features. Elimination techniques are again used to solve the system of partial derivative equations. One difference between the two localization algorithms is that resultant methods are used *twice* for the circular features case because the partial derivatives $\frac{\partial Error(X,Y,t)}{\partial X}$ and $\frac{\partial Error(X,Y,t)}{\partial Y}$ are nonlinear in X and Y.

4.5.1 Algorithm Overview

Resultant methods involve formulating the error function generically in terms of algebraic coefficients, formulating the partial derivatives in terms of the algebraic coefficients, and then constructing the symbolic resultant matrix. Resultant methods are used to eliminate X and Y from the system of partial derivative equations to produce an equation *solely in* θ, which can be solved numerically. Then, resultant methods are used again to eliminate X from a system of 2 partial differential equations to produce an equation solely in Y. Then, the remaining pose parameter X is determined for each orientation θ and y translation component Y.

There are five offline steps (similar to points and linear features):

1. Determine the structure of a generic error function by examining an algebraic expression of the error between an arbitrary transformed point and an arbitrary circular feature.

2. Express the error function $Error(X, Y, \theta)$ as a generic algebraic function in terms of symbolic coefficients.

3. Formulate the partial derivatives of the generic error function $Error(X, Y, \theta)$ with respect to the coordinates: X, Y, and θ. The motivation is that each zero-dimensional pose with local extremum error satisfies $\nabla \cdot Error(X, Y, \theta) = \vec{0}$.

4. Eliminate X and Y from the system of partial derivatives $\nabla \cdot Error(X, Y, \theta) = \vec{0}$ in order to generate a expression *solely in* θ. The result of this step is a symbolic resultant matrix which is used to solve for all of the θ of poses with local extrema error function. Given the orientation, θ, the remaining pose parameters X_θ, Y_θ can be determined by solving a system of linear equations.

5. Eliminate X from the system of two partial derivative equations $\frac{\partial Error(X,Y,\theta)}{\partial X} = 0$ and $\frac{\partial Error(X,Y,\theta)}{\partial Y} = 0$ to produce an equation *solely in* Y_θ, which can be solved numerically. The remaining pose parameter X is computed numerically using $\frac{\partial Error(X,Y,\theta)}{\partial X} = 0$.

In addition to the five offline steps, there are three online steps:

1. Instantiate the symbolic coefficients of the error function using the data set of points and associated linear features.

2. Compute all of the interesting orientations θ by solving the resultant matrix polynomial using eigenvalue techniques (refer section 4.4.3)

3. $\forall\ \hat{\theta}$, if $\hat{\theta}$ is a candidate orientation ($\mathrm{Im}(\hat{\theta}) \approx 0$):

   (a) Compute all of the *interesting* $\hat{Y_{\hat{\theta}}}$ by solving the resultant matrix polynomial using eigenvalue computations described in section 4.4.3.

   (b) $\forall\ \hat{Y_{\hat{\theta}}}$, if $\hat{Y_{\hat{\theta}}}$ is a candidate Y translation ($\mathrm{Im}(\hat{Y_{\hat{\theta}}}) \approx 0$):

      i. Compute for $X_{\hat{Y_{\hat{\theta}}},\hat{\theta}}$ using $\frac{\partial Error(X,Y,\theta)}{\partial X}|_{\hat{\theta},\hat{Y_{\hat{\theta}}}} = 0$ which is cubic in $X_{\hat{Y_{\hat{\theta}}},\hat{\theta}}$

      ii. Compute $Error(X_{\hat{Y_{\hat{\theta}}},\hat{\theta}}, \hat{Y_{\hat{\theta}}}, \hat{\theta})$ at each local extremum in order to find the global minimum error pose.

4.5.2 Approximating the Error Between Points and Circular Features

Unfortunately, resultant techniques cannot effectively minimize the squared error between points and circular features. In this section, we prove that there is no algebraic

rational polynomial expression of the point position (x, y) which describes the squared minimum distance between a point and a circle. This is shown by the following example: consider a point $(\vec{x,0})$ on the x-axis and circle C centered at the origin with radius R (see Figure 4.11(a)). The squared minimum distance between the point $(\vec{x,0})$ and the circle C is shown in Figure 4.11(b). The slope discontinuity of the squared minimum distance at $x = 0$ proves that no algebraic rational polynomial *in* x *alone* can describe the square of the minimum distance between a circle and a point.

The minimum distance between a point and a circle can be described by a modified system of algebraic polynomials by introducing an additional variable α denoting a point on the circle C. But introducing a new variable for each circular feature in that manner would prohibit the use of a single generic symbolic resultant matrix. Furthermore, introducing these extra degrees of freedom would increase the complexity (and therefore the running time) of the eigenvalue computations.

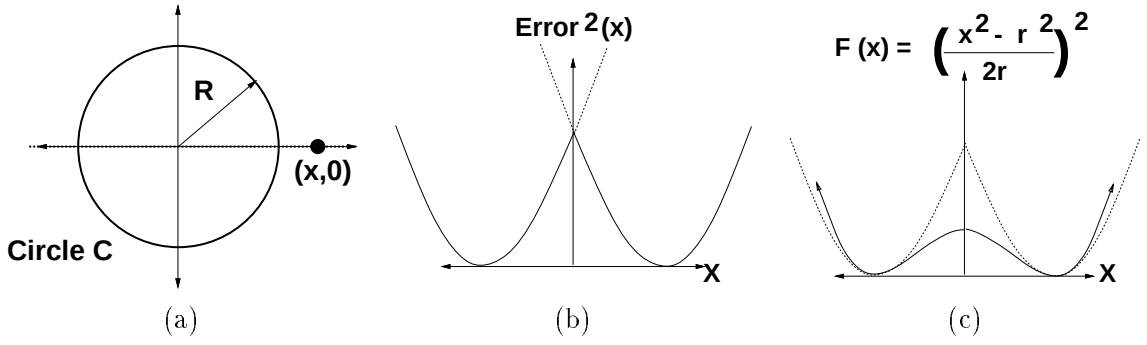


Figure 4.11: (a): Consider a point $(\vec{x,0})$ on the x-axis and a circle C centered at the origin of radius R. (b): The squared minimum distance between $(\vec{x,0})$ and C as a function of x. Notice the slope discontinuity at $x = 0$ which proves that the squared minimum distance between a point and a circle cannot be written as a algebraic rational polynomial function in the point position (x, y). (c): The function $F(x)$ approximates the squared minimum error for points nearby circular features.

For these reasons, we employ a rational polynomial function $F(x) = (\frac{x^2-r^2}{2r})^2$ to approximate the squared error function between points and circular features (see Figure 4.11(c)). This approximation should suffice when the data points are nearby the circular features.

4.5.3 Total Squared Error Function

In this section, we derive the squared error between points and circular features. We show that the squared error between a set of points and a set of linear and circular features can be expressed as a fourth order rational polynomial function in t.

$$\begin{aligned} Error_{total}(X,Y,\theta) &= Error_{linear\ features}(X,Y,\theta) + Error_{circular\ features}(X,Y,\theta) \quad (4.32)\\ &= \underbrace{\sum_i Error_{linear}(X,Y,\theta,\vec{x}_i,\vec{a}_i)}_{linear\ feature\ error} + \underbrace{\sum_i Error_{circular}(X,Y,\theta,\vec{x}_i,\vec{c}_i,r_i)}_{circular\ feature\ error} \end{aligned}$$

The error between a transformed point $\mathcal{T}(X,Y,t)\vec{x}$ and a circle C is exactly:

$$Error_{circular}(X,Y,t,\vec{x},\vec{c},r) = (\|(\mathcal{T}(X,Y,\theta)(\mathtt{X},\mathtt{Y},\mathtt{1})^{\mathsf{T}} - (\mathtt{c_x},\mathtt{c_y},\mathtt{1}))\| - \mathtt{r})^2 = \quad (4.33)$$

$$(\|\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} - \vec{c}\| - \mathtt{r})^2 \approx F(\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} - \vec{c}, r) = \quad (4.34)$$

$$(\frac{\|\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} - \vec{c}\|^2 - \mathtt{r}^2}{2r})^2 = \frac{(\|Mat(X,Y,t)\vec{x}^{\mathsf{T}} - (1+t^2)\vec{c}\|^2 - (1+t^2)\mathtt{r}^2)^2}{(1+t^2)^2(2r)^2} \quad (4.35)$$

$$Error_{total}(X,Y,\theta) = \underbrace{\sum_i Error_{linear}(X,Y,\theta,\vec{x}_i,\vec{a}_i)}_{linear\ feature\ error} + \underbrace{\sum_i Error_{circular}(X,Y,\theta,\vec{x}_i,\vec{c}_i,r_i)}_{circular\ feature\ error} \quad (4.36)$$

$$\approx \sum \|\vec{a}^{\mathsf{T}} \cdot (\mathcal{T}(X,Y,\theta)\vec{x})\|^2 + \sum (\frac{\|\mathcal{T}(X,Y,\theta)\vec{x}^{\mathsf{T}} - \vec{c}^{\mathsf{T}}\|^2 - \mathtt{r}}{2r})^2 \quad (4.37)$$

$$= \frac{\sum \|\vec{a}^{\mathsf{T}} \cdot (Mat(X,Y,t)\vec{x})\|^2}{(1+t^2)^2} + \frac{\sum \frac{1}{4r^2}(\|(Mat(X,Y,t)\vec{x} - (1+t^2)\vec{c})\|^2 - \mathtt{r}(1+t^2)^2)^2}{(1+t^2)^4} \quad (4.38)$$

It turns out that $\|(Mat(X,Y,t)\vec{x} \Leftrightarrow (1+t^2)\vec{c})\|^2 \Leftrightarrow \mathtt{r}(1+t^2)^2)^2$ is divisible by $(1+t^2)^2$ due to the fact that $\cos^2\theta + \sin^2\theta = 1$. By factoring $(1+t^2)^2$ out of the numerator and denominator of $Error_{circular}(X,Y,t)$, we arrive at a quartic polynomial expression. The linear feature error function and the factored circular feature error function have equal denominators ($(1+t^2)^2$) which enables the error functions to be easily added by adding the numerators of the error functions. The squared error between points and linear features was derived in section 4.4. We end up with an error function $Error_{total}(X,Y,t)$ which is a quartic function in t divided by $(1+t^2)^2$.

$$Error_{total}(X,Y,t) = \frac{\sum \|\vec{a} \cdot Mat(X,Y,t)\vec{x}\|^2}{(1+t^2)^2} + \frac{\sum \frac{1}{4r^2}(\frac{\|Mat(X,Y,t)\vec{x} - (1+t^2)\vec{c}\|^2}{(1+t^2)^2} - (1+t^2)r)^2}{(1+t^2)^2} \quad (4.39)$$

$$FF_{total}(X,Y,t) = Error_{total}(X,Y,t)(1+t^2)^2 \quad (4.40)$$

Once again, we use an algebraic function $FF_{total}(X,Y,t)$ to describe the sum squared error between points and linear and circular features, where $FF_{total}(X,Y,t) = Error_{total}(X,Y,t)$

(equation (4.41)). The partials $\frac{\partial Error_{total}(X,Y,t)}{\partial X}$, $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}$, $\frac{\partial Error_{total}(X,Y,t)}{\partial t}$, specify a system of equations defining the local extrema of the error function (including the global minimum).

$$\begin{aligned} FF_{total}(X,Y,t) = {} & \mathtt{a} + X\mathtt{ax1} + XY\mathtt{ax1y1} + XY^2\mathtt{ax1y2} + X^2\mathtt{ax2} + X^2Y\mathtt{ax2y1} + X^3\mathtt{ax3} \\ & + Y\mathtt{ay1} + Y^2\mathtt{ay2} + Y^3\mathtt{ay3} + t\mathtt{at1} + Xt\mathtt{ax1t1} + XYt\mathtt{ax1y1t1} + XY^2t\mathtt{ax1y2t1} + X^2t\mathtt{ax2t1} \\ & + X^2Yt\mathtt{ax2y1t1} + X^3t\mathtt{ax3t1} + Yt\mathtt{ay1t1} + Y^2t\mathtt{ay2t1} + Y^3t\mathtt{ay3t1} + t^2\mathtt{at2} \\ & + Xt^2\mathtt{ax1t2} + XYt^2\mathtt{ax1y1t2} + XY^2t^2\mathtt{ax1y2t2} + X^2t^2\mathtt{ax2t2} + X^2Yt^2\mathtt{ax2y1t2} \\ & + X^3t^2\mathtt{ax3t2} + Yt^2\mathtt{ay1t2} + Y^2t^2\mathtt{ay2t2} + Y^3t^2\mathtt{ay3t2} + t^3\mathtt{at3} + Xt^3\mathtt{ax1t3} \\ & + XYt^3\mathtt{ax1y1t3} + XY^2t^3\mathtt{ax1y2t3} + X^2t^3\mathtt{ax2t3} + X^2Yt^3\mathtt{ax2y1t3} + X^3t^3\mathtt{ax3t3} \\ & + Yt^3\mathtt{ay1t3} + Y^2t^3\mathtt{ay2t3} + Y^3t^3\mathtt{ay3t3} + t^4\mathtt{at4} + Xt^4\mathtt{ax1t4} + XYt^4\mathtt{ax1y1t4} \\ & + XY^2t^4\mathtt{ax1y2t4} + X^2t^4\mathtt{ax2t4} + X^2Yt^4\mathtt{ax2y1t4} + X^3t^4\mathtt{ax3t4} \\ & + Yt^4\mathtt{ay1t4} + Y^2t^4\mathtt{ay2t4} + Y^3t^4\mathtt{ay3t4} + (1+t^2)^2(X^2+Y^2)^2\mathtt{ax4y4} \end{aligned} \tag{4.41}$$

In the case of generalized-polygonal objects, some of the algebraic coefficients are redundant. We note these redundancies because they led to symbolic simplification.

ax3t2 = 2 ax3 - ay3t1	ax3t3 = ax3t1	ax3t4 = ax3 - ay3t1	ax1y2t1 = ax3t1
ay3t2 = 2 ay3 + ax3t1	ax1y2t3 = ax3t3	ax1y2t4 = ax3t4	ax1y2t2 = ax3t2
ay3t4 = ay3 + ax3t1	ay3t3 = ay3t1	ax2y1t1 = ay3t1	ax2y1t2 = ay3t2
ax2y1t3 = ay3t3	ax2y1t4 = ay3t4		

Again, the global minimum is found by finding all of the local extrema and finding the local extrema with minimum error. All of the local extrema are found by solving for simultaneous solutions to the zeros of the partial derivatives of the error function (refer equation 4.42).

$$\nabla \cdot Error_{total}(X,Y,\theta) = (0,0,0) \Rightarrow \nabla \cdot Error_{total}(X,Y,t) = (0,0,0) \tag{4.42}$$

Again, since we are only finding the common roots of equation (4.42), we are at liberty to use substitute any function $H_X(X,Y,\theta)$ for $\frac{\partial Error_{total}(X,Y,t)}{\partial X}$ with the proviso that $\frac{\partial Error_{total}(X,Y,t)}{\partial X} = 0 \rightarrow H_X(X,Y,\theta) = 0$. Such a substitution is necessary because we will be using resultant techniques which require algebraic (not rational) functions.

For $\frac{\partial Error_{total}(X,Y,t)}{\partial X}$ and $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}$, $H_X(X,Y,t)$ and $H_Y(X,Y,t)$ are equal to the partial derivatives $\frac{\partial FF_{total}(X,Y,t)}{\partial X}$, $\frac{\partial FF_{total}(X,Y,t)}{\partial Y}$. $H_t(X,Y,t)$ is slightly more complicated

because the denominator of $\frac{\partial Error_{total}(X,Y,t)}{\partial t}$ is a function of t, and this must be taken into account.

$$\begin{aligned}
\frac{\partial Error_{total}(X,Y,t)}{\partial X} &\propto \frac{\partial FF_{total}(X,Y,t)}{\partial X} \\
H_X(X,Y,t) &= \frac{\partial FF_{total}(X,Y,t)}{\partial X} \\
\frac{\partial Error_{total}(X,Y,t)}{\partial Y} &\propto \frac{\partial FF_{total}(X,Y,t)}{\partial Y} \\
H_Y(X,Y,t) &= \frac{\partial FF_{total}(X,Y,t)}{\partial Y} \\
\frac{\partial Error_{total}(X,Y,t)}{\partial t} &\propto \frac{\partial \frac{FF_{total}(X,Y,t)}{(1+t^2)^2}}{\partial t} = \frac{((1+t^2)\frac{\partial FF_{total}(X,Y,t)}{\partial t} - 4tFF_{total}(X,Y,t))}{(1+t^2)^3} \quad (4.43) \\
H_t(X,Y,t) &= ((1+t^2)\frac{\partial FF_{total}(X,Y,t)}{\partial t} - 4tFF_{total}(X,Y,t)) \quad (4.44)
\end{aligned}$$

The resultant of this system of equations is obtained by eliminating X and Y using the Macaulay construction [Mac02].

$$\begin{aligned}
M(t) &= Resultant(\{\frac{\partial Error_{total}(X,Y,t)}{\partial X}, \frac{\partial Error_{total}(X,Y,t)}{\partial Y}, \frac{\partial Error_{total}(X,Y,t)}{\partial t}\}, \{X,Y\}) \\
&= Resultant(\{H_X(X,Y,t), H_Y(X,Y,t), H_t(X,Y,t)\}, \{X,Y\}) \quad (4.45)
\end{aligned}$$

4.5.4 Solving for Orientation, t, of Local Extrema Poses

The resultant of the system of equations: $\nabla \cdot Error_{total}(X,Y,t) = 0$ is a 36×36 matrix with rank 34. Furthermore, we can eliminate two rank-deficient rows and many redundant singleton rows (rows containing only one non-zero element). Symbolic manipulation produces a 23×23 matrix with three singleton rows. The roots of the 23×23 matrix polynomial are computed using the eigendecomposition algorithms described in section 4.4.3.

The numerical precision of the resultant approach depends on the resultant matrix having full rank. Rank deficiency can result from *redundant* coefficients. The 26×26 construction we describe is designed for combinations of linear and circular features. If there are only circular features, we need to reexamine the symbolic resultant to eliminate any remaining rank deficiencies. For eigenvalue computations, rank deficiency results in numerical imprecision. For robustness, we generate both resultant formulations, one assuming linear features are included, and one assuming linear features are not included, and use the appropriate matrix to compute the orientation t.

4.5.5 Solving for Y Position, Y_t, of Local Extrema Poses

To compute Y_t after solving for t, we again use resultant methods since the remaining equations are nonlinear in X and Y. The resultant is constructed using the equations: $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ and $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t = 0$. At a particular orientation t, $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ can be written as a function $U(X_t, Y_t) = 0$, and $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t = 0$ can be written as a function $V(X_t, Y_t) = 0$. $U(X_t, Y_t) = 0$ is cubic in X_t and $V(X_t, Y_t) = 0$ is quadratic in X_t.

$$U(X_t, Y_t) = 0 \quad \Leftrightarrow \quad u_3(Y_t)X_t^3 + u_2(Y_t)X_t^2 + u_1(Y_t)X_t + u_0(Y_t) = 0 \tag{4.46}$$

$$V(X_t, Y_t) = 0 \quad \Leftrightarrow \quad v_2(Y_t)X_t^2 + v_1(Y_t)X_t + v_0(Y_t) = 0 \tag{4.47}$$

We use the Macaulay resultant to eliminate X_t and produce a function in Y_t (refer equation (4.48)) [Mac02]. This determinant of the resultant matrix is a third-order matrix polynomial in Y_t. This construction is more complex than the construction for the case of linear equations (refer section 4.4.2). We compute the roots of Y_t at simultaneous solutions using the eigenvalue decomposition method we previously outlined (see section 4.4.3).

$$Resultant(\{U(X_t, Y_t), V(X_t, Y_t)\}, X_t) = \begin{bmatrix} u_0(Y_t) & 0 & v_0(Y_t) & 0 & 0 \\ u_1(Y_t) & u_0(Y_t) & v_1(Y_t) & v_0(Y_t) & 0 \\ u_2(Y_t) & u_1(Y_t) & v_2(Y_t) & v_1(Y_t) & v_0(Y_t) \\ u_3(Y_t) & u_2(Y_t) & 0 & v_2(Y_t) & v_1(Y_t) \\ 0 & u_3(Y_t) & 0 & 0 & v_2(Y_t) \end{bmatrix} \tag{4.48}$$

4.5.6 Solving for X Position, $X_{Y_t,t}$,of Local Extrema Poses

Given Y_t and t, $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ is a cubic equation in $X_{Y_t,t}$ which can be solved numerically. $\frac{\partial Error_{total}(X,Y,t)}{\partial X}|_t = 0$ is used, rather than $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t = 0$ (which is quadratic in $X_{Y_t,t}$), in case $\frac{\partial Error_{total}(X,Y,t)}{\partial Y}|_t$ is independent of $X_{Y_t,t}$.

4.5.7 Verifying (X_{Y_t}, Y_t, t) is a Local Extremum

Resultant methods can produce extraneous solutions because they are necessary but not sufficient condition. This particular method can generate $26 \times 4 \times 5 \times 3$ configurations. Once again, we need to verify that each (X_{Y_t}, Y_t, t) configuration is a local extremum by computing the absolute values of the partial derivatives and comparing them to some ϵ threshold which accommodates sensor noise and numerical error.

4.5.8 Example

In this example, consider the case of the four points corresponding to two circles as shown in Figure 4.12. One pose which exactly maps the four points onto their corresponding features is given in Figure 4.13. The four points are: $(\Leftrightarrow 2, 3), (\Leftrightarrow 1, 2), (\Leftrightarrow 2, 1), (\Leftrightarrow 2, \Leftrightarrow 3)$, and the two associated circles are $(x + 2)^2 + y^2 = 1, (x \Leftrightarrow 2)^2 + y^2 = 1$; the first four points all correspond to the first circle.

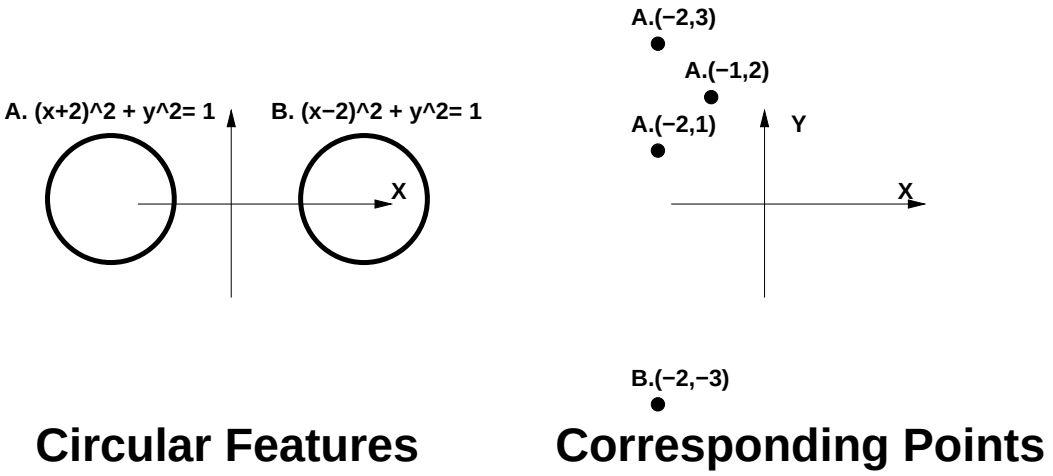


Figure 4.12: Four points corresponding to two circular features

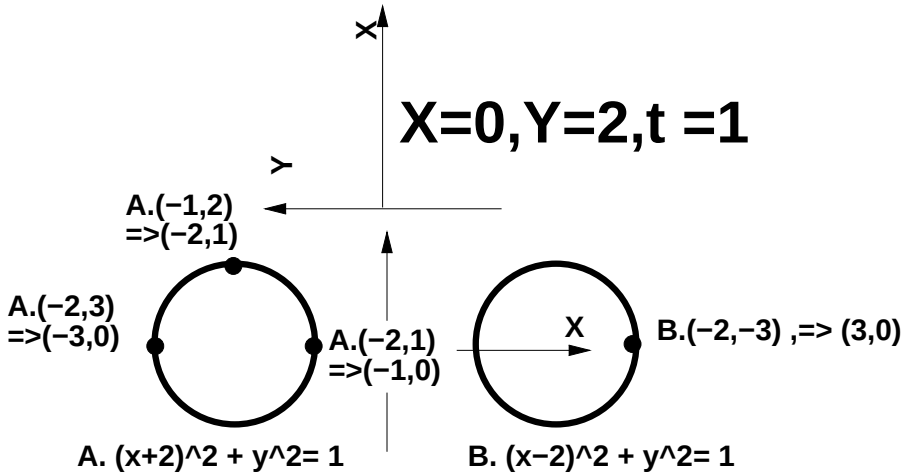


Figure 4.13: One solution transformation which maps the four points onto the corresponding circular features.

The total error function between the transformed points and the associated circular

features is given in Figure 4.14 as a function of y and t. The global minimum of the error function is zero at $\theta = \frac{\pi}{2}$, $Y = 2$, which corresponds to mapping the vertices directly onto the corresponding circular features. The algorithm found two distinct local minima, the global minima and also $X = \Leftrightarrow 2.1629, Y = 0.0915, \theta = \pi)$.

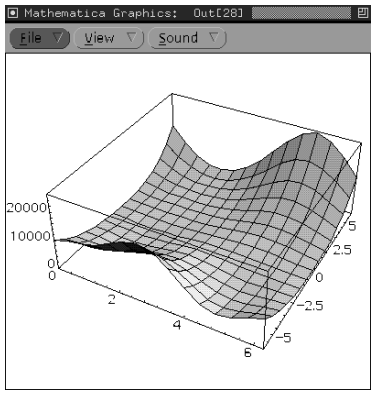


Figure 4.14: The total error function $Error(Y, \theta)$ for the four points and associated circular features.

4.6 Implementation and Performance

In this section, we describe the performance of the localization algorithm on many cases. We have applied the algorithm on randomly generated data for large data sets as well as investigated the relationship between camera resolution and accuracy using simulated data. We used the localization algorithm to compare the localization performance of using all available boundary information as compared to just utilizing isolated boundary points, and we experimentally tested the technique for small data sets using real data.

In the first case, we generated random sensed data, consisting of random features and then random feature points. To simulate the random positions, all of the feature points were transformed by a random rigid two-dimensional transformation. Finally, we ran the localization on the random features and transformed feature points and compared the estimated transformation with the actual transformation.

For the second suite of experiments, we investigated the relationship between pixel resolution and localization accuracy by localizing objects in known poses at different resolutions. We began with simple polygonal models representing the objects' boundaries, and moved the models into random positions: x, y, θ. For each pose and resolution, we enumer-

ated all of the pixel squares covered by the boundary features and listed these corresponding feature(s) with each pixel. Finally, we compared the estimated transformations with the actual transformations for different pixel resolutions to understand the implications of camera resolution.

4.6.1 Verifying the Technique With Random Features

Generating Random Point and Linear Feature Data

Point and corresponding linear feature data (edges) were synthesized as shown in Figure 4.15:

1. Random Linear Feature (Edge) Generation:
   (a) Randomly choose the edge's direction from a uniform distribution $[0 \ldots 2\pi]$.
   (b) Randomly choose the edge's distance from the origin from a uniform distribution $[min_{distance} \ldots max_{distance}]$.
2. Random Point Sampling from Linear Feature (Edge):
   (a) Construct a circle centered at the origin of random radius by randomly choosing a radius r from a uniform distribution $[min_{radius} \ldots max_{radius}]$.
   (b) Find the point on the edge by intersecting the edge with the circle formed in step (a).
3. Random Gaussian Perturbation of Data Point:
   (a) Randomly choose a perturbation to the point, by choosing the orientation and length of the perturbation: orientation from a uniform distribution $[0 \ldots 2\pi]$, length from normal distribution with mean 0 and standard deviation σ.

min_{length} was 0.0, max_{length} was 10.0, min_{radius} was 10.0, max_{radius} was 15.0, and σ was 0.1.

Generating Random Point, Circular Feature Data

Point and corresponding linear feature data and corresponding circular feature data were synthesized by generating point and linear feature data as described in section 4.6.1, and generating point and circular feature data as shown in Figure 4.16:

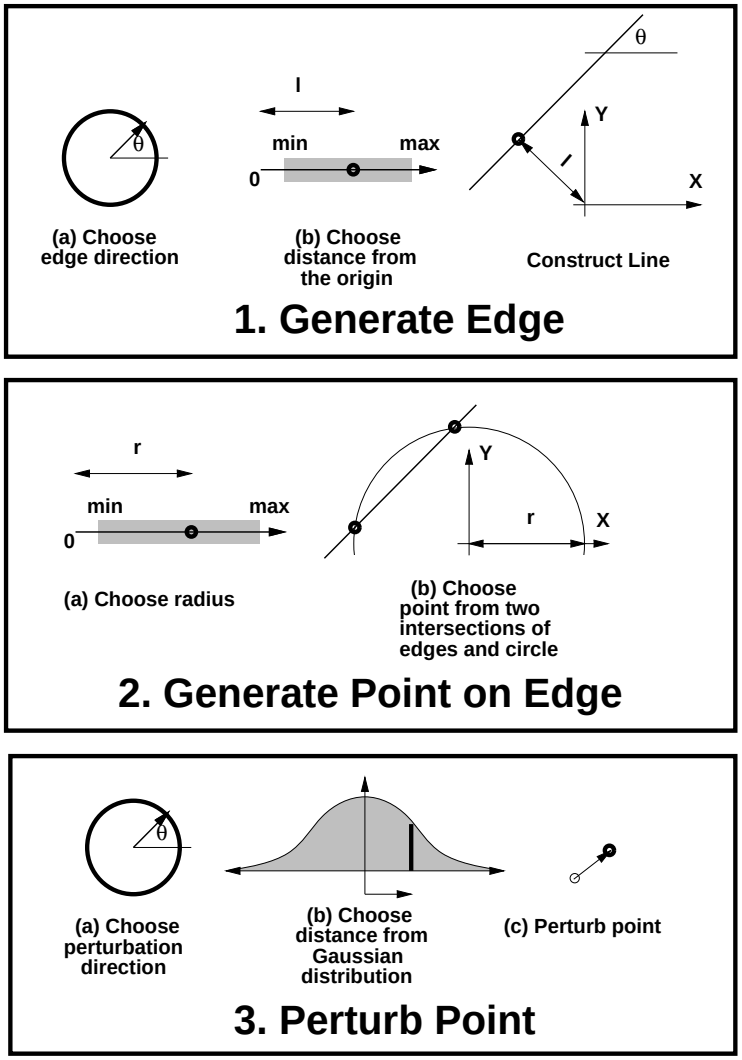


Figure 4.15: Method for constructing random point and linear feature data

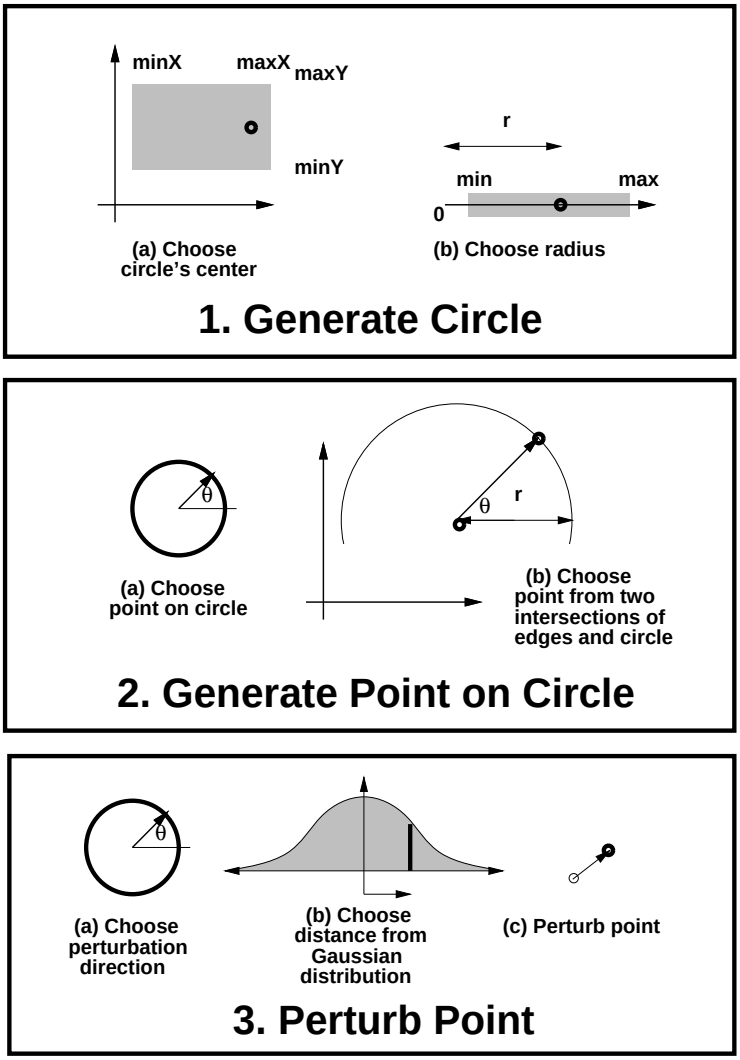


Figure 4.16: Method for constructing random point and circular feature data

1. Random Circular Feature Generation:

   (a) Randomly choose the center of the circle (from a square) by choosing random x and y positions from uniform distributions: $[min_{coordinate} \ldots max_{coordinate}]$.

   (b) Randomly choose the circle's radius r from a uniform distribution: $[min_{radius} \ldots max_{radius}]$.

2. Random Point Sampling from Circular Feature:

   (a) Randomly choose the point on the circle perimeter from a uniform distribution $[0 \ldots 2\pi]$.

3. Random Gaussian Perturbation of Data Point:

   (a) Randomly choose a perturbation to the point by choosing the orientation and length of the perturbation: orientation from a uniform distribution $[0 \ldots 2\pi]$, length from normal distribution with mean 0 and standard deviation σ.

$min_{coordinate}$ was -10.0, $max_{coordinatetest}$ was 10.0, min_{radius} was 3.0, max_{radius} was 15.0, and σ was 0.1.

Results

Table 4.8 compares the estimated poses with the actual poses for the randomly generated data sets of points and linear features and circular with perfect sensing ($\sigma = 0.0$). Table 4.9 compares the estimated poses with the actual poses for the randomly generated data sets of points and linear features with $\sigma = 0.1$. Table 4.10 compares the estimated poses with the actual poses for the randomly generated data sets of points and linear and circular features with $\sigma = 0.1$.

4.6.2 Relationship Between Pixel Resolution and Localization Accuracy

Most of the work in machine vision is camera-based and involves high precision, high data bandwidth sensors. This localization algorithm is useful for edge-detection based approaches because it enables the user to utilize all the edge data. In regards to edge-detection based approaches, we wanted to investigate the relationship between camera resolution and localization accuracy.

Random Point and Linear and Circular Feature Data			
# linear features	# circular features	Actual pose	Estimated pose
8	0	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.0, Y = 2.0, \theta = 0.7)$
8	0	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.0, Y = 2.0, \theta = 0.8)$
8	0	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.0, Y = 2.0, \theta = 0.9)$
8	0	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.0, Y = 6.0, \theta = 1.0)$
4	4	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.0, Y = 2.0, \theta = 0.7)$
4	4	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.0, Y = 2.0, \theta = 0.8)$
4	4	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.0, Y = 2.0, \theta = 0.9)$
4	4	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.0, Y = 6.0, \theta = 1.0)$
0	8	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.0, Y = 2.0, \theta = 0.7)$
0	8	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.0, Y = 2.0, \theta = 0.8)$
0	8	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.0, Y = 2.0, \theta = 0.9)$
0	8	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.0, Y = 6.0, \theta = 1.0)$

Table 4.8: Performance of (point,linear and circular feature) localization technique for randomly generated data with $\sigma = 0$.

Random Point and Only Linear Feature Data		
# of random data points	Actual pose	Estimated pose
100000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 1.998597, Y = 2.003240, \theta = 0.700061)$
100000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.001527, Y = 2.002663, \theta = 0.800075)$
100000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.001533, Y = 2.002073, \theta = 0.900087)$
100000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 3.998578, Y = 6.001493, \theta = 1.000099)$
1000000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 1.999183, Y = 2.000571, \theta = 0.699966)$
1000000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 3.000826, Y = 2.000516, \theta = 0.799959)$
1000000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 1.000823, Y = 2.000461, \theta = 0.899951)$
1000000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 3.999190, Y = 6.000408, \theta = 0.999945)$

Table 4.9: Performance of (point,linear feature) localization technique for randomly generated data for $\sigma = 0.1$.

Random Point and Linear and Circular Feature Data			
# linear features	# circular features	Actual pose	Estimated pose
100000	100000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.000344, Y = 2.001496, \theta = 0.699958)$
100000	100000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 2.999460, Y = 2.001097, \theta = 0.799954)$
100000	100000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 0.999193, Y = 2.000738, \theta = 0.899951)$
100000	100000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.001135, Y = 6.00432, \theta = 0.999948)$
1000000	1000000	$(X = 2, Y = 2, \theta = 0.7)$	$(X = 2.000105, Y = 1.999681, \theta = 0.699999)$
1000000	1000000	$(X = \Leftrightarrow 3, Y = 2, \theta = 0.8)$	$(X = \Leftrightarrow 2.999905, Y = 1.999641, \theta = 0.799999)$
1000000	1000000	$(X = \Leftrightarrow 1, Y = 2, \theta = 0.9)$	$(X = \Leftrightarrow 0.999903, Y = 1.999603, \theta = 0.900003)$
1000000	1000000	$(X = 4, Y = 6, \theta = 1.0)$	$(X = 4.000110, Y = 5.999568, \theta = 1.000005)$

Table 4.10: Performance of (point,linear and circular features) localization technique for randomly generated data $\sigma = 0.1$.

There are many factors which affect the quality of localization estimates, such as lens distortion, lighting, surface reflectance, etc. In these experiments, we concentrated only on the relationship between pixel resolution and localization accuracy in order to gain a clear understanding. The term pixel refers to an individual sensor in the sensor matrix. We assume that the resolution of the data produced by the camera is limited by the pixel size. The term pixel resolution refers to the area in the scene (in the neighborhood of a particular object) characterized by a single pixel.

We believe that many systems rely on comparing isolated points on boundary curves, such as vertices and inflection points, because there are easily reproducible published algorithms for efficiently solving this problem. The purpose of these experiments is to determine the how significantly the localization precision improves by utilizing all of the available data, rather than only isolated boundary points. We wanted to determine whether utilizing all of the edge data was significantly better compared to using only isolated points on boundary curves; it was intuitive to us that using more of the available data always improves performance. We found a distinct advantage to using all of the available information in that our localization technique provided not insignificantly better estimates.

The experiments consisted of localizing a known object from a library of four modeled objects in a known pose, and recording the variation between the actual and estimated pose parameters. These experiments involved generating simulated data for polygonal models in multiple poses for multiple camera resolutions. We used simulated data in order to focus on the relationship between pixel resolution and localization accuracy. We assumed a

model of square pixels. The data points were of the form: $\langle$ *pixel center, associated linear feature* $\rangle$, and the data was synthesized by enumerated the centers of all pixel squares which overlapped any of the modeled boundary features (refer Figure 4.17). A picture of the pixelization of the hex-model object is shown in Figure 4.19.

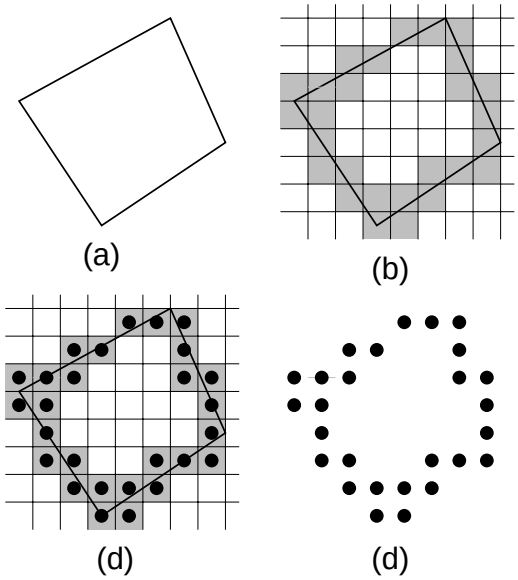


Figure 4.17: Simulated data was synthesized by enumerating the centers of all of the square pixels crossed by the boundary features

Four different polygonal models were used, and each polygonal model was localized in ten different poses at five different pixel resolutions. In order to characterize the results, we measured the errors in x, y, θ in a manner similar to standard deviations. In other words, we summed up the squares of the discrepancies between the actual and estimated pose parameters for x, y, θ, and computed the square root of the average. To characterize the average error, we simply averaged the errors returned by the optimal fit returned by the localization technique. These error measurements are reported in Table 4.11.

The results from using only data points corresponding to the vertices are shown in Table 4.12. The most important variable (for polygons) is θ because the x and y pose parameters are linear functions and can be easily solved given θ. Notice that the pose parameter estimates decrease by a factor between three and four when the resolution is doubled. At very high resolutions, increasing the resolution has little effect on the x, y, θ pose estimates; for most of the objects, the pose parameter estimates x, y converge to within 0.01 units of the actual translation (0.02% of the object length), and the orientational

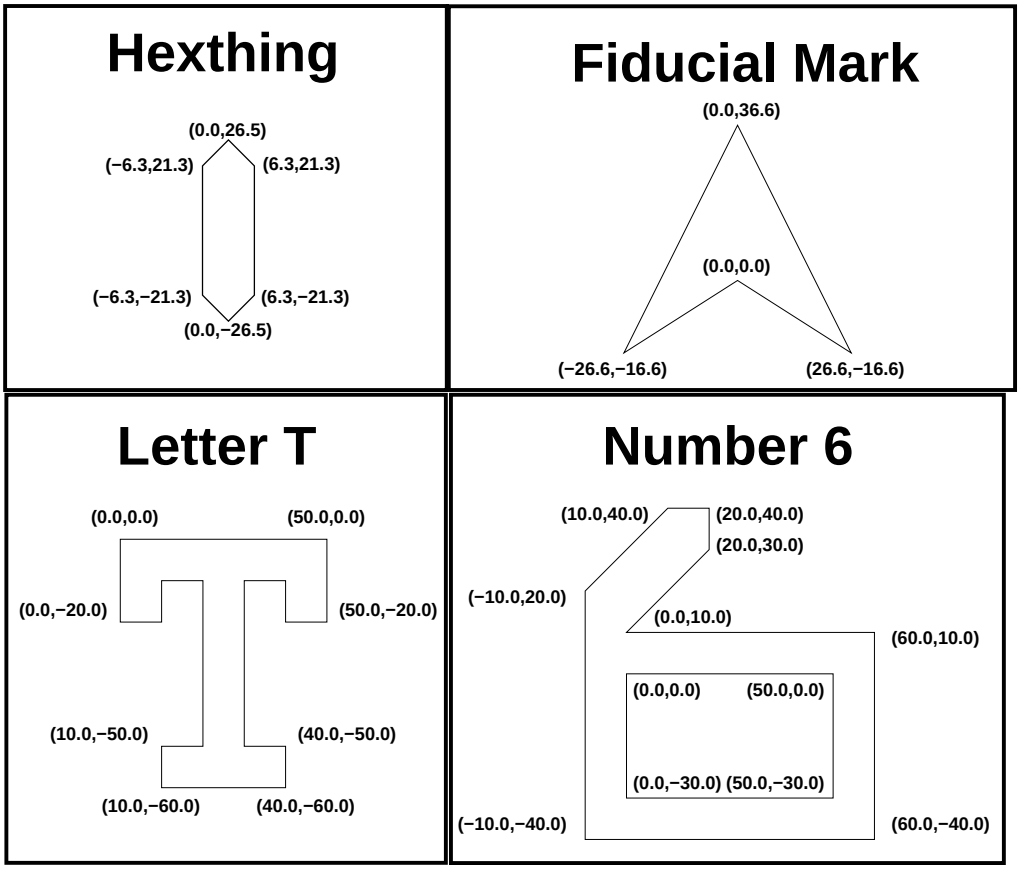


Figure 4.18: Four objects: Hex-model, Fiducial, Letter T, Number 6. Almost all of the vertex positions are given, but including the vertex positions for the letter T would clutter the figure without adding any additional information

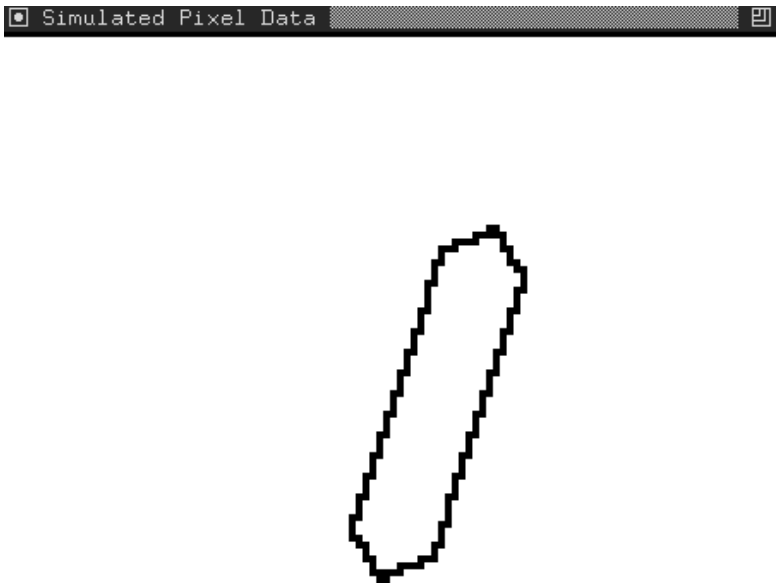


Figure 4.19: The simulated pixel data for the hex-model object at position $(7, \Leftrightarrow 3)$ at orientation -0.32 radians at resolution 1 pixel unit

Object	Resolution	|Pixels|	$\sqrt{\frac{1}{n}\sum_n (x \Leftrightarrow \hat{x})^2}$	$\sqrt{\frac{1}{n}\sum_n (y \Leftrightarrow \hat{y})^2}$	$\sqrt{\frac{1}{n}\sum_n (\theta \Leftrightarrow \hat{\theta})^2}$	$\overline{Error}$
Hex-model	4.0	44.4	0.33446434	0.23129888	0.013330161	92.927956
Hex-model	2.0	82.4	0.07352662	0.049923003	0.0047521484	44.196507
Hex-model	1.0	157.6	0.041777074	0.0063066096	0.0036297794	21.544956
Hex-model	0.5	309.5	0.0031298357	0.0016034179	4.643562e-4	10.784967
Hex-model	0.25	614.6	0.0042416616	3.8894132e-4	1.0569962e-4	5.4200945
Letter "T"	4.0	49.2	0.5133025	0.60952026	0.021071866	103.38039
Letter "T"	2.0	90.4	0.13007967	0.25665605	0.00581296	48.95305
Letter "T"	1.0	174.3	0.054255202	0.124452725	0.0047051026	24.013884
Letter "T"	0.5	343.2	0.012289527	0.017529113	6.203042e-4	12.062334
Letter "T"	0.25	679.6	0.0040623806	0.0034328252	1.9236103e-4	6.0177937
Number "6"	4.0	106.5	0.21361086	0.19730452	0.010275254	228.42712
Number "6"	2.0	205.8	0.054341987	0.049577143	0.004476396	112.03862
Number "6"	1.0	405.9	0.010724932	0.02340998	0.0010456733	56.430653
Number "6"	0.5	803.6	0.0059675984	0.004836324	1.6956787e-4	28.311207
Number "6"	0.25	1600.9	0.0017914316	0.0019320602	7.470778e-5	14.09613
Fiducial	4.0	61.4	0.21314956	0.14251679	0.007586037	128.73416
Fiducial	2.0	120.0	0.057874706	0.07354973	0.0030514833	66.32908
Fiducial	1.0	237.4	0.029797971	0.02263394	8.284851e-4	33.25131
Fiducial	0.5	468.0	0.004916789	0.004921388	1.2238462e-4	16.175451
Fiducial	0.25	932.4	0.007628871	0.0050956924	1.399027e-4	8.207249

Table 4.11: Average localization technique performance for various objects at various resolutions

Object	Resolution	|Pixels|	$\sqrt{\frac{1}{n}\sum_n (x \Leftrightarrow \hat{x})^2}$	$\sqrt{\frac{1}{n}\sum_n (y \Leftrightarrow \hat{y})^2}$	$\sqrt{\frac{1}{n}\sum_n (\theta \Leftrightarrow \hat{\theta})^2}$	$\overline{Error}$
Hex-model Corners	4.0	12.2	0.8087669	0.74312437	0.021346115	11.319004
Hex-model Corners	2.0	12.5	0.15569122	0.25277224	0.013486578	3.3304203
Hex-model Corners	1.0	13.1	0.036884286	0.116834834	0.00448842	1.0500076
Hex-model Corners	0.5	14.9	0.024275968	0.024562975	0.005042185	0.26821047
Hex-model Corners	0.25	17.4	0.0063632606	0.004727341	0.0024016364	0.09157471

Table 4.12: Average localization technique performance using only the corners of the hex-model object at various resolutions

parameter estimate θ converges to a precision of 0.0001 radians. Note that the sum squared error is halved when the resolution is doubled, implying that the squared error per datapoint shrinks by a factor of four, since there are approximately twice as many data points at the finer resolution.

Next, we investigated the performance degradation related to only utilizing isolated boundary points, rather than all of the available boundary information. To accomplish this, we generated data points We did not utilize the published algorithms for estimating pose from point sets, but instead used the localization technique with only the data points from vertices. All such data points are easily found by modifying the object model, and then synthesizing the data in exactly the same manner. The model was modified by replacing each edge by extremely small edges at the endpoints. In this way, the data synthesizer only generates data points for the vertices. This approach does not give the exact same answer as using the commonly used point set algorithm because the error function of the localization technique is not the squared distance between the transformed model vertex and the actual vertex, but the sum of the two squared errors between the transformed actual vertices and the two linear model features. The statistics for using only the vertices for the hex-model object are given in Table 4.12.

We interpret these results from Table 4.12 in comparison to localization experiments using only the corner data. The most important variable (for polygons) is θ because the x and y pose parameters are linear functions and can be easily solved given θ. At the sharpest resolution, the θ estimates for the localization technique are 20 times more accu-

rate than the vertex-based localization technique. Furthermore, errors in x, y are larger for the vertex case at every resolution save one.

The only error statistic which does not differ significantly between the two approaches is the squared error. On a per-data point basis, the squared error of the localization technique at the sharpest resolution $\frac{5.42}{614.6}$ (approximately 0.01) is very similar to the squared error per data point of the vertex based method, $\frac{0.09}{17}$. We found that the pose estimates were more accurate using more data points, which is an intuitive result. It is confusing to note that the sum squared errors are relatively equivalent; one explanation is that since there are fewer data points, a vertex-based approach more easily fits the data, even though it does not produce a more precise result.

4.7 Conclusion

4.7.1 Conclusion

In this chapter, we described an object localization technique for sensor and model features of different types, such as polygonal models, and probe points. The approach involved reducing the pose estimation problem to a least squares error problem. We utilized algebraic elimination techniques to exactly solve the least squares error problems. The main advantages of this localization algorithm are its applicability both to sparse and dense sensing strategies, its immunity to local minima problems, its exact computation of the global minimum, and its high speed. We used this technique to compare the relative performance of using only a subset of the data (interesting features such as vertices), with using the entire data set, and we found the intuitive result that utilizing more data provides a more precise position estimate. This algorithm has been successfully applied it for recognition and localization applications using very precise optical sensors.

4.7.2 Future Work

This approach is extendible to three-dimensional localization from three-dimensional range data. This would involve computing generic error functions between points and features of co-dimension one (planes, surfaces) as a function of the six degrees of freedom. After the generic error functions have been determined, all we need to do is construct larger resultant matrices to solve for the minima. The resultant solving may take a slightly longer

time owing to the greater number of degrees of freedom and algebraic complexity. This approach may also be extendible to three-dimensional localization from two-dimensional perspective image data. In this case, we can consider the image points to be rays going through the eye, and compute the disparity between these rays and model edges in three dimensions. The crucial step involves succinctly formulating the generic error between pairs algebraically. Again, resultant solving may take a longer time, owing to the greater number of degrees of freedom and algebraic complexity.

Chapter 5

Planning for Modular and Hybrid Fixtures

Abstract

Fixturing encompasses the design and assembly of fixtures to locate and hold a workpiece during a manufacturing operation such as drilling or assembly. Automated fixture design is very difficult because an infinite variety of fixtures is possible. Fixture quality depends not only on the configuration and the expected process operations, but also upon the mechanical characteristics of the fixturing elements in the fixture kit. We believe that a fixture kit consisting of simple modular mechanical components can be more precise, less expensive, and more robust than a fixture kit of complicated components. We have implemented automated fixture design algorithms for a modular fixturing kit consisting of a fixture vise and modular fixturing elements (pegs or flatted pegs). Generally, an appropriately sized fixture vise system allows many configurations for immobilizing a prismatic workpiece. In this chapter, we present an algorithm which accepts as input a model of a workpiece and enumerates all force closure fixtures, where the term "force closure" describes the capability of the fixture to resist arbitrary forces and torques. The algorithm runs in $O(A)$ time, where A is the number of configurations achieving simultaneous contact between the modeled object and four fixture elements; since simultaneous contact is a necessary but not sufficient precondition for force closure, A provides an upper bound on the number of force closure fixture configurations.

5.1 Introduction

In this chapter, we describe an algorithm which enumerates all fixture vise configurations for immobilizing a given prismatic object. The running time of the enumeration algorithm depends upon the number of valid configurations. The fixture vise system consists of two fixture plates mounted on jaws of a vice and reconfigurable modular fixturing elements. Researchers have tried to implement automated fixture design algorithms for other modular fixture kits [AB85; Shi93; CCB89; MMFF84], but such attempts have been hampered by the sheer complexity of the task: most fixture kits have excessive degrees of freedom (more than the degrees of constraint), and this results in infinite numbers of valid fixtures for most workpieces. The fixture vise, on the other hand, has exactly the same number of degrees of freedom as number of degrees of constraint, yielding a finite number of valid fixture configurations. We focus on the fixture vise because a complete enumeration algorithm would be impossible without this finite bound.

As Hazen and Wright [HW90] noted:

> ...tooling engineers have endeavored to give [jigs and fixtures] flexibility and modularity so that they can be applied to the greatest possible set of part styles. Such flexibility is even more important today, since the trend in manufacturing is toward production in small batch sizes[Mil85]. Batch production represents 50 to 75 percent of all manufacturing, with 85 percent of the batches consisting of fewer than 50 pieces[GTG88]. As the batch size for a particular part decreases, modularizing fixtures and jigs can help to minimize the setup costs per unit produced.

and as Gandhi and Thompson [GT86] noted:

> With the [Flexible Manufacturing Systems] projected to achieve an annual expenditure of $1.8 billion by the year 1990, and the typical cost of dedicated fixtures amounting to 10-20% of the total systems costs, the technological and economical impact of flexible fixturing technologies will clearly be of major significance. Several examples in the available literature clearly demonstrate that adaptable fixturing will reduce these costs by 80%.

Smaller batches and higher tolerances are hallmarks of the current trend in manufacturing. Flexible manufacturing addresses many of these issues, including part variety, high quality, high reliability and rapid changeover, and it requires a wide variety of fixture shapes.

The task of immobilizing and locating a workpiece via mechanical devices is commonly called fixturing or workholding, and its automation poses difficult questions, such as: *How do task constraints, material, etc. influence fixture quality? What effect does the expected batch size have on the fixture quality metric? How should dimensional tolerances be treated? etc.*

By definition, fixtures must repeatably constrain the object's location and provide force closure, the ability to resist arbitrary forces and torques. The expected forces range from 20KN for machining fixtures down to 50N for assembly fixtures. An automated fixture design system is greatly desired for efficiency.

However, research on analytical fixture design has proceeded slowly because fixture design is currently more of a craft than a science. Due to the craft-like semblance of fixture design, many automated fixture design systems are based on knowledge engineering or expert systems; the state of the art commercial systems are CAD libraries of modular fixturing sets. Such tools are a far cry from interactive automated fixture design systems. An automated fixture design system tailored to a particular modular toolkit would not only increase productivity, but could also impact part design.

Even though expert systems and the CAD libraries can perform the rudimentary task of verifying force closure, such tools lack a firm analytical foundation: CAD libraries allow virtual experimentation, but cannot synthesize fixture designs, and usually do not even analyze designs. Without geometric analysis, an expert system can only describe "types" of fixturing components; it cannot propose fixture designs. Although recent efforts have combined rule-based heuristics with geometrical placement heuristics, these systems are incomplete, in that they cannot generate the infinite variety of valid fixture designs. In this chapter, we describe a complete geometrically-based fixture design system.

Without constraints or limitations, the problem of designing fixtures currently seems too open ended to be solved in a reasonable amount of time. As a first step towards automated fixture design, we analyze the task of enumerating all frictionless point contact fixture vise configurations for immobilizing prismatic objects, which can alternatively be described as objects of constant cross-section. Fixtures achieving force closure under a frictionless point contact assumption are usually preferred over fixtures assuming frictional contact or soft fingers because large stresses and aberrations can result from very large resistive frictional forces.

The fixture vise is akin to both fixture tables, plates inlaid with precision drilled

holes, and the Black & Decker Workmate system, a fixturing system including peg holes in two vise jaws. A fixture vise system consists of modular fixture elements (pegs or flatted pegs) inserted into fixture tables mounted on both jaws of a vise (Figure 5.1). For the remainder of this chapter, we will use the term pegs to denote modular fixturing elements. The term configuration characterizes the state of the fixture vise system; *i.e.,* the peg positions and the separation between the vise jaws.

The fixture vise possesses the minimum number of degrees of freedom necessary (one) to deal with workpiece variations. We conjecture that any noncircular prismatic object can be fixtured using an appropriately sized fixture vise system. Consider the following thought experiment: configure the fixture vise so that there are two pegs on the innermost columns on the bottom two rows; separate the jaws as wide as necessary to accommodate an arbitrary object (which contains in its interior a disc of radius δ, the spacing between each pair of pegs); compress the jaws together; assume there is zero friction between the pegs and the object so that the object does not become jammed; if the object does not leave the rectangle defined by the four pegs, then the system will eventually converge to a stable equilibrium which will either involve three or four contacts. It seems unlikely that all stable equilibrium configurations should involve only three contacts.

The fixture vise promises wide applicability, high quality, and easy reconfiguration. The fixture vise can immobilize a significant class of workpieces: prismatic workpieces. Its simple design enables high precision tooling and easy reconfiguration due to the peg/hole mechanism. Moreover, our complete fixture design algorithm enables the fixture vise to rapidly handle different workpieces. A fixture vise system can also be viewed as a *general-purpose reconfigurable* gripper when turned upside down.

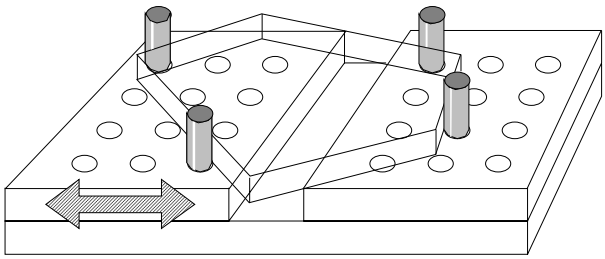


Figure 5.1: A fixture vise consists of two fixture table jaws which translate along one direction.

The fixture vise system is a reasonable tradeoff between genericity and realizability,

flexibility and efficiency, computational complexity and utility; it is able to perform useful fixturing tasks, but at the same time, it is also amenable to automated fixture design.

The computational tractability is due to the reduction of a three dimensional, six degree of freedom fixturing problem to a two dimensional, three degree of freedom fixturing problem. We only need to consider the two-dimensional projection of the scene since we are immobilizing constant cross-section objects using constant cross-section pegs (Figure 5.2).

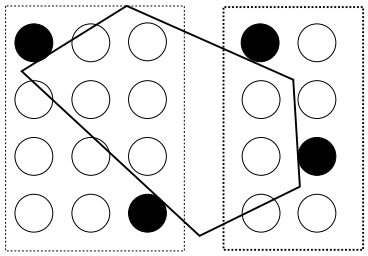


Figure 5.2: The two-dimensional projection of the object and fixture vise from Figure 1.

The fixture design algorithm is basically an enumeration algorithm which exploits problem-specific pruning heuristics. Without pruning heuristics, the routine would compute configurations for every quartet of peg contacts and every quartet of edges, and then check these configurations for force closure. We implemented these heuristics using an elegant parameterization which simplifies the constraint representations.

It is desirable to enumerate all possible fixture configurations because of the variety of criteria for ranking fixtures, such as : minimizing the maximal forces required to counter any unit force through the object's center of mass, minimizing the maximal forces required to counter torques along a specific cutting path, etc. One problem with relying on heuristics to find an *optimal* design is that it hinders the design process by restricting combinations of criteria. This approach could be just as efficient and yet more flexible than simple heuristic based methods by using heuristics to guide the enumeration algorithm. More important is the prospect of finding a single fixture configuration which can immobilize a series of workpieces to lessen changeover time.

Next, we will discuss the advantages of using a fixture vise system, and then depict an interactive fixture design session, followed by a discussion of the related work.

5.1.1 Advantages of the Fixture Vise System

The fixture vise has two advantages over an ordinary vise: the fixture vise's ability to handle arbitrary face contacts, and the fixture vise's ability to utilize internal contacts. Unlike an ordinary vise which can only contact prismatic objects at parallel faces or edges, fixture vise elements can contact prismatic objects at arbitrary faces, enabling fixture vises to immobilize objects without crushing corner edges. Fixture vises can also immobilize objects using internal faces of holes in the object, while ordinary vises cannot.

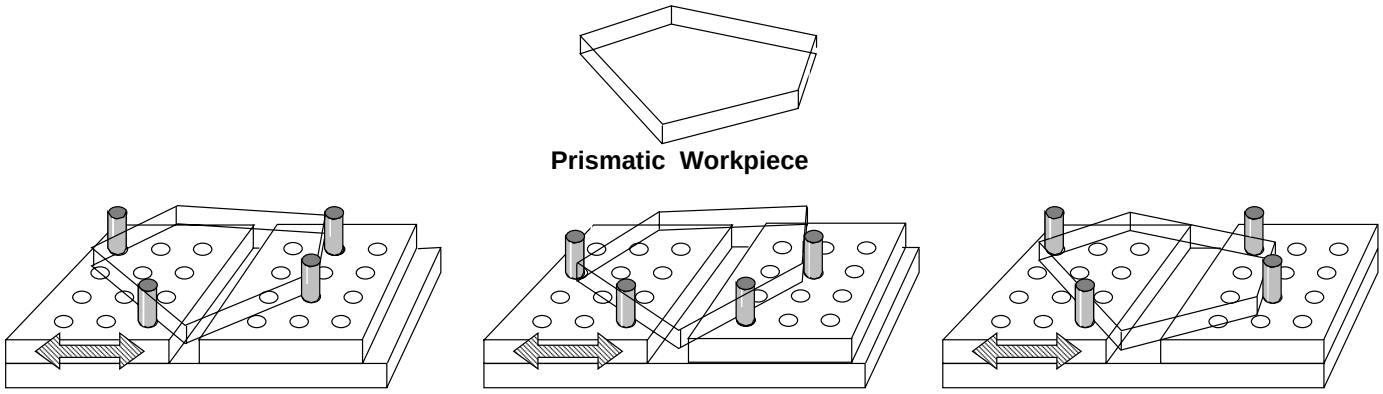


Figure 5.3: Three fixture vise configurations for immobilizing a given prismatic object.

But the most important aspect of the fixture vise system is the discrete fixel placements which allow a combinatorial search algorithm (Figure 5.3). Usually, many different fixture vise configurations are suitable, even more so if different radii fixture element pegs are available. Figure 5.3 shows a number of configurations capable of fixturing a single part, and Figure 5.4 shows different objects immobilized using the same fixture vise configuration. Finding a configuration which can immobilize a variety of workpieces requires considering all configurations. Thereby, fixture vise systems can reduce changeover time by utilizing *slightly suboptimal configurations* (in terms of some metric) for which the pegs remain in the same configuration for fixturing a series of workpieces.

5.1.2 Interactive Fixture Design

Before we describe our algorithm for enumerating all of the fixture vise configurations, we first describe an interactive fixture design session. Initially, the operator has selected a workpiece, and has conceptualized the manufacturing operations for producing

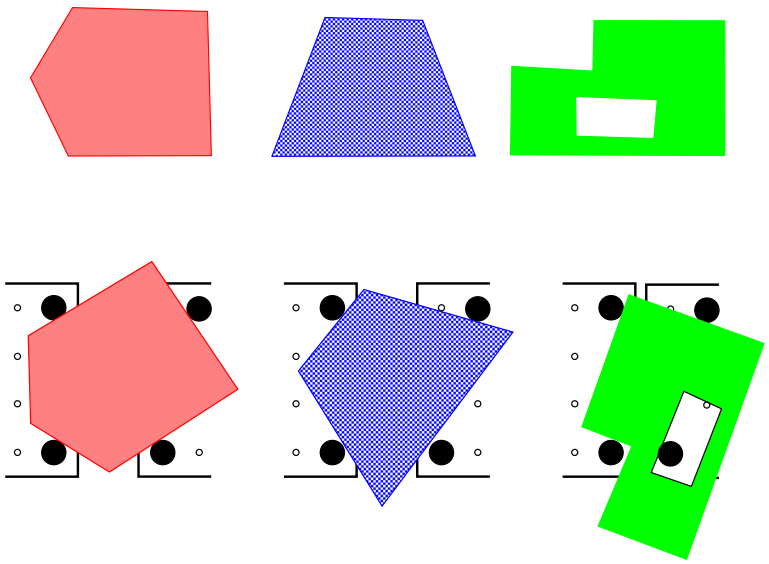


Figure 5.4: Multiple objects can be fixtured using the same fixture vise configuration.

the finished product (Figure 5.5). The fixture design algorithm performs step 2: computing valid fixture vise designs.

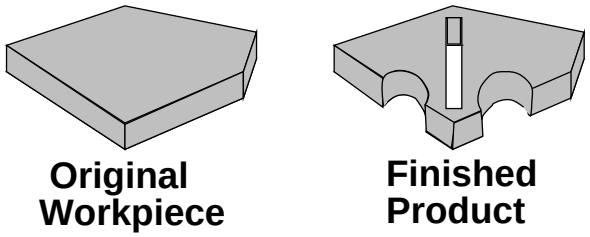


Figure 5.5: Initially, the operator selects a workpiece and conceptualizes the manufacturing operations for transforming the workpiece into finished product.

1. The operator selects a two-dimensional model of the projection of the workpiece and highlights the areas of the model requiring accessibility (Figure 5.6).

2. The computer program determines all possible fixture vise configurations for the workpiece satisfying the accessibility constraints. If no such fixture vise configuration exists, then the algorithm enumerates sequences of fixture vise configurations which, combined, satisfy the accessibility constraints.

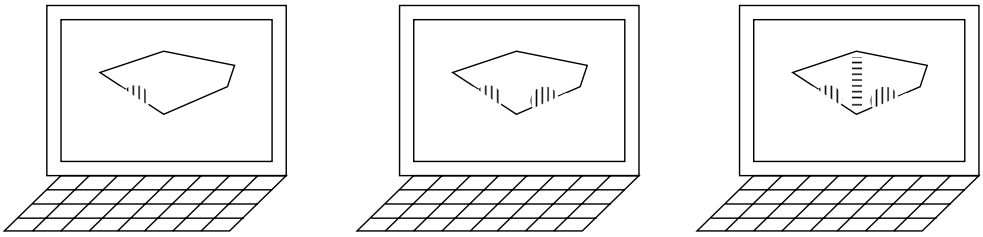


Figure 5.6: The operator selects a two-dimensional model of the projection of the workpiece and highlights the areas of the model requiring accessibility.

3. After selecting one of the fixture configurations, the operator sets up the fixture vise by placing the appropriate (flatted) pegs in appropriate holes.

4. The operator places the workpiece approximately in the fixture and tightens the vise, squeezing the workpiece into its appropriate pose (Figure 5.7).

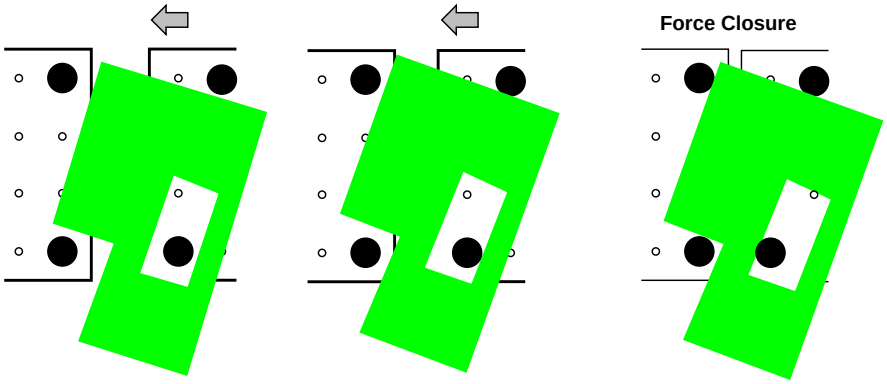


Figure 5.7: Closing the vise squeezes the workpiece into its appropriate pose.

5. The operator verifies that the workpiece is fixtured either manually, by trying to twist the workpiece, or visually, by checking that the appropriate faces of the workpiece contact the appropriate fixture elements, and that the workpiece's position is approximately correct.

6. The operator performs some or all of the the manufacturing operations depending upon whether refixturing is necessary.

After describing the interactive fixture design process, we proceed to discuss the related work in detail.

5.1.3 Related Work

Since fixturing is a fundamental task in assembly, its automation is greatly desired in industry. Automated fixture design systems should increase the designer's productivity by performing rudimentary design tasks, possibly even suggesting improvements. Some fixture design systems even generate code to robotically assemble the designs [AB85]. This research is related to work on Reduced Intricacy Sensing and Control (RISC) Robotics promoted by John Canny at U.C. Berkeley and Ken Goldberg at U.S.C. [CG94] (section 2.1.1).

Due to their significance, fixturing and fixture design have been the focus of many research projects. Fixtures and automated fixture design are discussed at length in the mechanical engineering literature and the manufacturing literature. Fixture design is also related to work done on grasping by the robotics community.

Towards analying fixtures, Reuleaux [Reu76] defined the term force closure to be: a set of contacts on an object which can resist the force of gravity. Nguyen redefined the term force closure to be: a set of contacts on an object capable of resisting arbitrary forces and torques. Most of the work in fixturing and grasping has been based on the notion of a *wrench* introduced by Ohwovoriole [Ohw87], which refers to both the forces and associated torques.

Asada and By were the first to utilize computers to design and synthesize fixtures for generic objects. Asada and By described the modular *Automatically Reconfigurable Fixturing (ARF)* system [AB85], analyzed the kinematics of object immobilization, and also presented accessibility and detachability conditions to guarantee that the operator can insert and extract the workpiece into and out of the fixture.

Hazen and Wright reviewed the mechanical engineering literature on automated fixturing techniques, components, planning, and execution [HW90]. Their review included many novel fixturing systems, such as phase change materials, in which the part is placed in liquified material which is then solidified. Shirinzadeh [Shi93] described a family of specialized one degree support and clamp modular fixture elements adhered to a magnetic chuck. These extra degrees of freedom allow an infinite number of fixtures, which poses the very difficult question of how to generate candidate designs.

Chou *et al.* presented an optimization technique for positioning the clamping elements of a fixture [CCB89]. Chou *et al.* analyzed the force torque fixture constraints for rectangular and polyhedral objects. Their technique output a sequence of clamp positions

which minimized the expected forces and torques.

Hazen and Wright's review [HW90] implied that most of the research in automated fixturing systems are based on expert systems. Markus *et al.* described an interactive fixturing design system based on expert systems which utilized towers, adjustable height fixturing elements embedded in a modular base [MMFF84]. Gandhi and Thompson [GT86] attempted to bridge the gap between geometrical analysis and expert systems via a technique for analyzing force closure within the framework of an expert system. Hayes and Wright [HW89] developed *Machinist*, an expert-system based nonlinear high-level process and fixturing planner.

Chang [Cha92] brought up a number of difficult problems usually ignored by automated fixture design systems: process planning and fixture planning are intertwined and should not be decoupled; the production type, the tool configuration, and the tolerances are all essential for evaluating fixture designs [Cha92].

There is renewed interest in the academic robotics community to work on industrial problems, such as automated fixture design. Mishra studied the problem of designing toe-clamp fixtures for rectilinear objects, and proved upper bounds on the number of toe clamps necessary to immobilize rectilinear polyhedral objects [Mis86].

In addition to the contributions from the mechanical engineering and manufacturing communities, the robotics community has also made significant contributions in terms of grasping. There are three main avenues of grasping research: existence proofs of grasps and lower bounds on the number of fingers for a force closure grasp, efficiently enumerating useful grasps, and metrics for evaluating grasp quality.

Many robotics researchers have worked on the problems of the minimum number of fingers necessary to achieve a force closure grasp of an object with and without friction. Fixturing aimed at manufacturing tasks is fundamentally different from grasp planning aimed at robotic tasks in that robotic grasping often exploits friction to achieve force closure, whereas manufacturing fixtures are often designed to achieve force closure without assuming friction, due to the very large and inherently dynamic forces. There are many exceptions such as circles, or surfaces of revolution, cannot be fixtured assuming frictionless point contacts, but are nonetheless fixtured every day.

Laksminarayana [Lak78] showed that seven frictionless point contacts are necessary to achieve force closure for rotationally asymmetrical three dimensional objects. Mishra *et al.* [MSS87] proved that seven fingers was an upper bound on the number of fingers required

for force closure for objects which are rotationally asymmetric. Markenscoff *et al.* proved lower bounds on the number of fingers necessary for immobilizing a two or three dimensional object with or without friction [MNP90].

There are many algorithms which enumerate useful grasps. Nguyen specified independent regions of contact on polygonal and polyhedral objects such that simultaneous contact on the object in each of the regions guaranteed force closure [Ngu88], and he presented a polynomial-time algorithm for enumerating all such sets of independent regions for polygons and polyhedra. Faverjon and Ponce extended Nguyen's work on independent regions to curved two-dimensional objects [FP91]. Markenschoff and Papdimitriou [MP89a] described an efficient algorithm for computing finger positions on polygonal objects which minimized the maximum force necessary to counter any unit force through the object's center of mass. Brost [Bro88] precomputed the orientations of the parallel-jaw gripper which would guarantee a successful grasping operation. Mirtich and Canny [MC94] presented a geometrically-based algorithm for computing optimal grasps for two dimensional and three dimensional objects.

Knowing the desired grasp, the next task is to achieve that grasp in the presence of uncertainty. Brost [Bro91] described the *backprojection* problem, the problem of determining initial states and plans which achieve a goal state under quasi-static assumptions. He described algorithms for computing such backprojections. Such results are relevant to the fixture vise in that we can utilize these analyses to determine the set of initial configurations for which a given object will rotate into the desired configuration.

Various criteria have been suggested to evaluate grasp quality. Li and Sastry [LS88] described kinematically-based metrics for evaluating grasps, and stressed the importance of considering the task by developing quantitative task-based metrics. Markenscoff and Papadimitriou [MP89a] measured grasp quality by the maximum force necessary to counter any unit force through the object's center of mass. Bausch and Youcef-Toumi presented a kinematically-based metric based on motion stops [BYT90]. Ferrari and Canny [FC92] introduced two grasp quality measures corresponding to maximum finger force and total finger force. More specifically for fixture design, Chou *at.* [CCB89] and Brost and Goldberg [BG94] verified that the fixture elements would not experience excessive forces from the expected operational forces.

Almost simultaneously Brost and Goldberg developed a different, though similar, modular fixturing system consisting of fixture elements and a side clamp mounted on a

fixture plate. Their algorithm also enumerates fixture designs for prismatic polyhedral objects [BG94]. In their paper, they discuss many of the same issues as in ours, as well as a more thorough discussion of quality metrics. Zhuang *et al.* proved that the fixturing system of Brost and Goldberg is *incomplete* by demonstrating a constructing class of large polygons which cannot be fixtured [ZGW94]. There have yet to be found any polygons or classes of polygons which cannot be fixtured using an appropriately sized fixture vice toolkit. Furthermore, all of translating clamp fixtures correspond to one subclass of fixture vice configurations, with the translating clamp corresponding to the jaw separating when one of the fixture plates contains only a single peg.

5.1.4 Overview

In section two, we present the theoretical background, and in section three we present an outline of the algorithm. We detail the two main subroutines in sections four and five. In section six, we present a few fixture vise configurations automatically designed for various object models, and then conclude by highlighting the results and advantages of this technique. This article elaborates on the fixture vise system introduced by Wallack and Canny [WC94] in Chapter 5.

5.1.5 Results

The main result of this work is a complete algorithm which enumerates all of the fixture designs suitable for immobilizing a prismatic object, in that the planar modular vise constrains planar object motions. This algorithm's significance lies in its efficiency, because it only generates simultaneous contact fixture configurations in which the object contacts all four fixture elements simultaneously (four is the minimum number of point contacts necessary to achieve force closure).

5.2 Theoretical Background

In this section, we present five ideas which lay the foundation for the algorithm: two observations, an argument which shows we can ignore a certain kind of degenerate case, a computational model of force closure, an argument stating that achieving four simultaneous contacts necessarily requires a total of four degrees of freedom, and a conservative test

for checking if a quartet of edges can admit a force closure configuration.

5.2.1 Observations

To simplify the analysis, we reduce the problem of achieving simultaneous contact with pegs to the problem of achieving simultaneous contact with points by appropriately transforming the object. Cylindrical or flatted pegs can conservatively be considered point contacts (pegs of zero radius) by transforming the edge segments (Figure 5.8). For circular pegs, this reduction is equivalent, but for flatted pegs, this reduction is conservative. This point contact approximation provides a consistent framework for treating both types of contacts. The drawback of this assumption is that we require four pegs per configuration, even though three flatted pegs could achieve force closure.

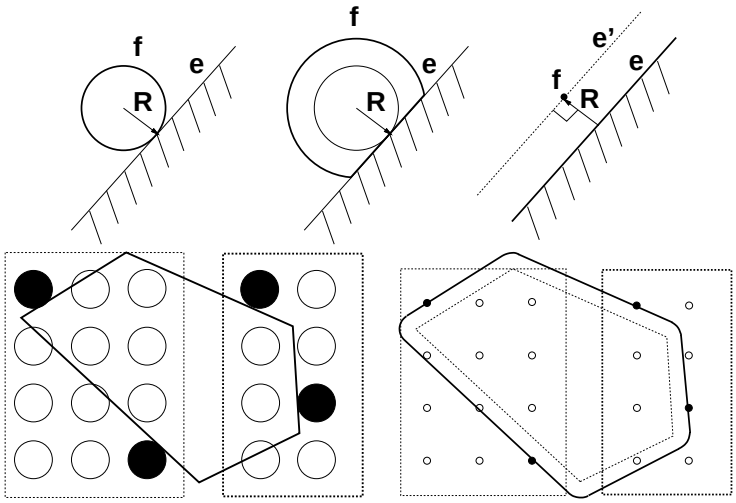


Figure 5.8: Cylindrical or flatted pegs can be considered point contacts by suitably shifting the corresponding edge.

Second, there are only two generic modes of simultaneous contact: Type I- two edge segments of the object contact pegs on each each jaw, and Type II- three of the object's edge segments contact pegs on a single jaw (Figure 5.9). We restrict our attention to peg/edge contacts since peg/vertex contacts would more likely incur part deformation, thereby reducing the predictability of part position.

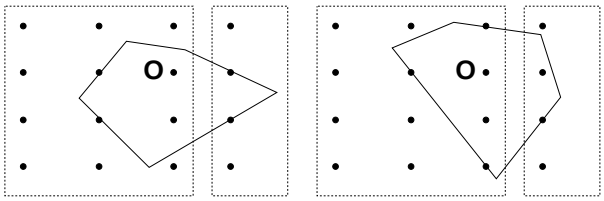


Figure 5.9: Type I (left): two edge segments of the object contact pegs on each jaw. Type II (right): three edge segments of the object contact pegs on a single jaw.

5.2.2 Parallel Edge Pair Case is Irrelevant

In this section, we show that a certain class of fixture configurations will never provide repeatable localization, and therefore can be ignored. We focus on this class in particular because it is a degenerate case for our pose parameterization; by proving that such cases are irrelevant, we do not need to provide a separate parameterization for these degenerate cases. The degenerate case we focus on is the case of Type I contact where the two edges contacting pegs on the left jaw are parallel and the two edges contacting pegs on the right jaw are parallel. The reason that we can ignore this case is because the fixture does not provide repeatable positioning of a workpiece.

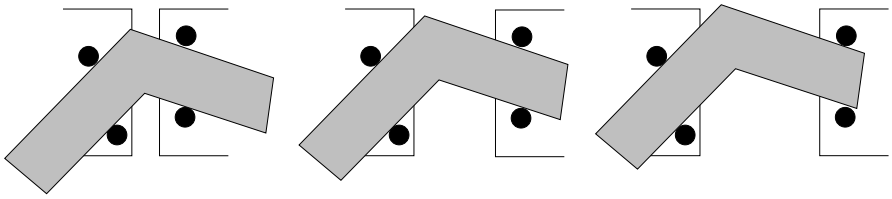


Figure 5.10: The case where each jaw contacts two parallel edges. We ignore this case because the jaw separation is not uniquely determined.

5.2.3 Verifying Force Closure Mathematically

In this section, we describe a novel mathematical method to check for force closure in the plane. Consider an object constrained by four point contacts (Figure 5.11). Forces at the four contact points induce four torques relative to an arbitrary reference point on the object. The torques correspond to the cross-products of the forces f with the lever arms r.

$$\tau = f \times r = f_x r_y \Leftrightarrow f_y r_x \tag{5.1}$$

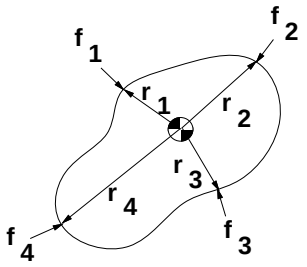


Figure 5.11: Forces at the four contact points induce four torques relative to a reference point on the object.

Force closure in two dimensions, given four point contacts, can be verified with matrices. Consider the four x, y, τ point contact vectors: $(f_{1,x}, f_{1,y}, \tau_1)^{\mathsf{T}}$, $(f_{2,x}, f_{2,y}, \tau_2)^{\mathsf{T}}$, $(f_{3,x}, f_{3,y}, \tau_3)^{\mathsf{T}}$, $(f_{4,x}, f_{4,y}, \tau_4)^{\mathsf{T}}$. We can construct a four by four matrix M consisting of the four contact vectors and four symbolic coefficients α_1, α_2, α_3, α_4:

$$M = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \\ f_{1,x} & f_{2,x} & f_{3,x} & f_{4,x} \\ f_{1,y} & f_{2,y} & f_{3,y} & f_{4,y} \\ \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{bmatrix} \tag{5.2}$$

The four vectors positively span the three-dimensional x, y, θ force torque space **if and only if** the minors of α_i in M all have the same sign (and none of the minors is zero).

5.2.4 Force Closure Requires At Least Four Degrees of Freedom

Fixtures for prismatic objects require at least four degrees of freedom. This is shown by a dimension counting argument: force closure requires four simultaneous contacts, and simultaneously satisfying four constraints generically requires at least four degrees of freedom. Rigid transforms of two dimensional objects provide only three degrees of freedom (x, y, θ). The fixturing system must provide at least one additional degree of freedom. In the fixture vise system, the degree of freedom is the separation σ.

5.2.5 Testing Quartets of Edge Segments for Possible Force Closure Contacts

Using linear programming, we can quickly test whether point contacts on four edge segments can possibly achieve force closure, and thereby we can prune unqualified

edge quartets. In wrench space, the contacts at all of the edge segment endpoints specify a polyhedron with the property that force closure cannot be achieved by point contacts on those edge segments (Figure 5.12) if the polyhedron does not contain a ball around the origin. Furthermore, this test suggests a heuristic to enumerate the edge quartets which have better chances of achieving useful fixtures. The edge quartets could be ordered by the volumes of these polyhedra.

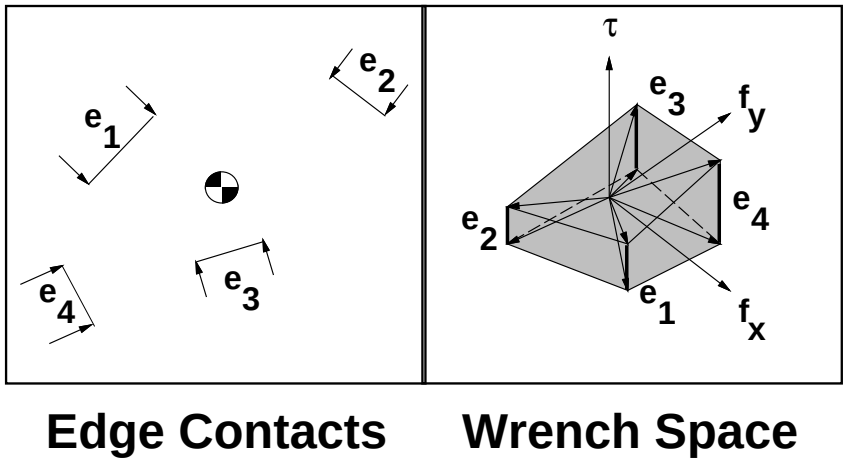


Figure 5.12: We can quickly test if force closure can be achieved for a particular quartet of edges by checking if a ball around the origin lies within the polyhedron

5.3 Algorithm Outline

In this section, we present an overview of the main algorithm which enumerates all fixture configurations via a generate-and-test strategy. We begin by defining the terms, followed by a specification of the algorithm's input and output. Then, we explain both the generate and test subroutines in more detail.

5.3.1 Overview

The *generate* subroutine generates all configurations providing simultaneous contact between (object model) edge segments and fixture pegs, and the *test* subroutine predicts the object poses corresponding to simultaneous contact and then tests for force closure.

5.3.2 Definitions

Definition 5.1 *Configuration* specifies the object's pose (X, Y, θ), the positions of the pegs $\vec{\mathcal{F}}$, and the jaw separation distance σ.

Definition 5.2 e refers to an edge segment, and $\vec{E}$ refers to an ordered quartet of edge segments $\vec{E} = (e_1, e_2, e_3, e_4)$.

Definition 5.3 $\vec{\mathcal{E}}$ refers to a quartet of *jaw-specified* edge segment where each of the edges is assumed to contact a peg on the left jaw or on the right jaw, *i.e.*, $\vec{\mathcal{E}} = (e_1 \times s_1, e_2 \times s_2, e_3 \times s_3, e_4 \times s_4), s_i \in \{left, right\}$.

Definition 5.4 $\vec{\mathcal{F}}$ refers to a quartet of positions of the peg centers contacting edge segments $\vec{\mathcal{E}}$. The pegs have been transformed down to points by suitably translating the edge segments.

5.3.3 Specification

The algorithm expects as input: a two dimensional polygonal model of a workpiece, a subset of edge segments representing the accessible boundary areas, a list of available peg radii, the sizes of the fixture plates mounted on the vise, and the maximum separation between the jaw plates. The algorithm provides as output a list of fixture configurations including the peg positions, part pose, and separation.

5.3.4 Generate and Test Subroutines

Now, we explain both the generate and test subroutines in more detail; the generate subroutine is sketched in Figures 5.13 and 5.15.

1. **Generate:**

   (a) Enumerate all jaw-specified edge segment quartets (combinations of four edge segments such that any edge segment can appear more than once) $\{\vec{E}, \vec{E}', \vec{E}'', \ldots\}$ for which we can possibly achieve force closure with four point contacts (section 5.2.5).

   (b) For each quartet of jaw-unspecified edge segments $\vec{E}$, enumerate all different combinations of jaw-specified contacts $\{\vec{\mathcal{E}}, \vec{\mathcal{E}}', \vec{\mathcal{E}}'', \ldots\}$. There are seven different

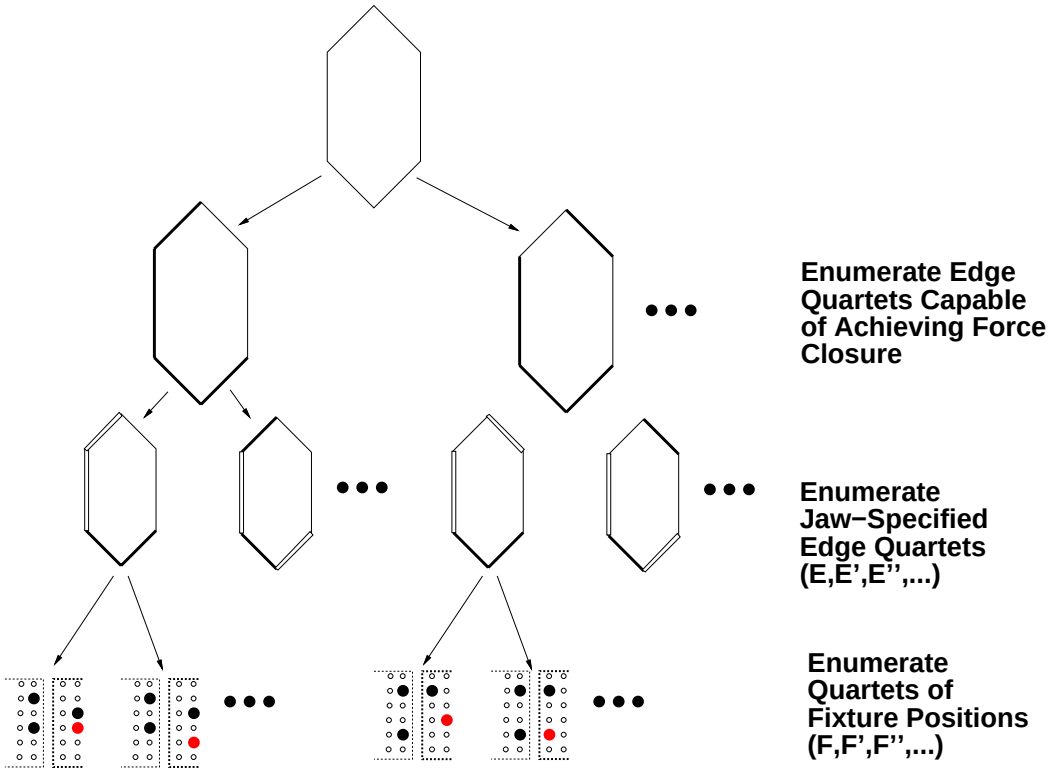


Figure 5.13: The generate algorithm enumerates quartets of jaw specified edge segments capable of producing force closure, and then, for each edge quartet, enumerates quartets of fixture positions (filled-in circles represent the fixel positions).

situations to be considered: four where three edge segments contact the left jaws, and three where pairs of edge segments contact both jaws.

(c) For each edge segment quartet $\vec{\mathcal{E}}$, compute fixture configurations $\{\vec{\mathcal{F}}, \vec{\mathcal{F}}', \vec{\mathcal{F}}'', \ldots\}$ providing simultaneous contact $\vec{\mathcal{E}}$ (Figure 5.14). The pegs are considered points via a suitable transformation of the corresponding edge segments. The fixture

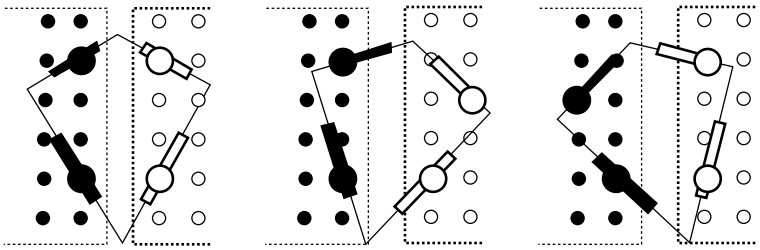


Figure 5.14: Different peg configurations $\vec{\mathcal{F}}$, $\vec{\mathcal{F}}'$, $\vec{\mathcal{F}}''$ simultaneously contacting edge segments $\vec{\mathcal{E}}$.

vise configurations $\{\vec{\mathcal{F}}, \vec{\mathcal{F}}', \vec{\mathcal{F}}'', \ldots\}$ are enumerated in the manner shown in Figure 5.15. We assume that the position of the first peg is the origin, then we enumerate all of the positions for the second peg. For each position of the second peg, we enumerate all of the positions of the third peg. Then we enumerate all of the positions of the fourth peg given the first three (section 5.4).

2. **Test:**

   (a) Compute the object poses and jaw separations achieving simultaneous contact between edge segments $\vec{\mathcal{E}}$ and peg positions $\vec{\mathcal{F}}$.

   (b) Verify force closure for each simultaneous contact pose.

5.3.5 Notation

- The operator $\oplus$ refers to the Minkowski sum which can be defined as: $A \oplus B = \{a + b | a \in A, b \in B\}$; we use the Minkowski to map from $S^1 \times S^1 \to S^1$ and $\mathbb{R}^1 \times \mathbb{R}^1 \to \mathbb{R}^1$

- λ_x and λ_y refer to the row and column spacing of the holes inlaid in the fixture tables on the vise jaws (Figure 5.16).

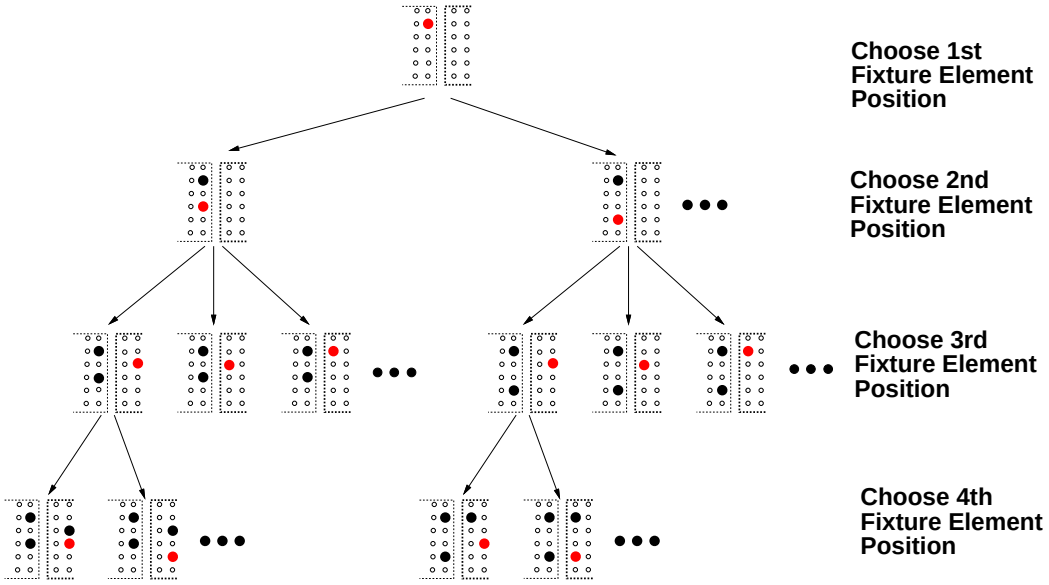


Figure 5.15: The fixture vise configurations are constructed by first choosing the position of the second peg, then choosing the position of the third peg and so on (filled-in circles represent the fixel positions).

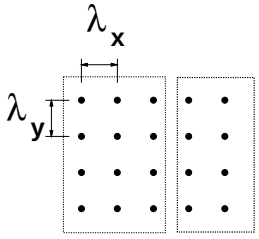


Figure 5.16: λ_y and λ_x refer to the spacing between the rows and columns for the fixture plates mounted on the jaws of the fixture vise.

5.4 Enumerating Fixture Configurations Contacting a Quartet of Edge Segments

5.4.1 Algorithm Outline

This subroutine efficiently enumerates all fixture configurations $\{\vec{\mathcal{F}}, \vec{\mathcal{F}}', \vec{\mathcal{F}}'', \ldots\}$ achieving simultaneous contact with a quartet of edge segments $\vec{\mathcal{E}}$. Our approach utilizes geometric constraints to enumerate A configurations in $O(A)$ time. Enumerating configurations using simpler heuristics would generate extraneous fixture configurations, and could increase the running time significantly.

Since the configurations are characterized by the relative peg positions, we are free to assume that the first peg is placed at the origin on the left jaw. The second peg position must lie within an annulus defined by geometric constraints. Enumerating the positions of the third and fourth pegs is more difficult than enumerating the positions of the second peg because the third and fourth pegs (for Type I contacts) reside on the right vise jaw which translates freely relative to the origin on the left vise jaw. The third (fourth) peg must lie in the region swept over by the third (fourth) edge segment which maintain contact between the first two edge segments and the first and second fixture pegs (Figure 5.17).

For Type I configurations, the third and fourth fixture elements reside on the right jaw which can translate freely in x (Figure 5.18). We compute the regions swept over by the third and fourth edges with respect to the origin, which provides relative information about the possible x positions of the third and fourth fixture pegs, and absolute positional information about the possible y positions.

Because of the lattice structure, we do not need to characterize the entire regions swept by those edge segments; we only need to determine which horizontal rows are swept over by the edge segments, and then determine the range of contiguous x values along each row.

Since the object only has three degrees of freedom (x, y, θ), the object's pose can be characterized by a single variable, after the first two peg positions are chosen. The pose is parameterized by χ, which corresponds to the orientation of the extended intersection (section 5.4.2). This parameterization is used because it simplifies the task of computing the regions swept over by the third and fourth edges.

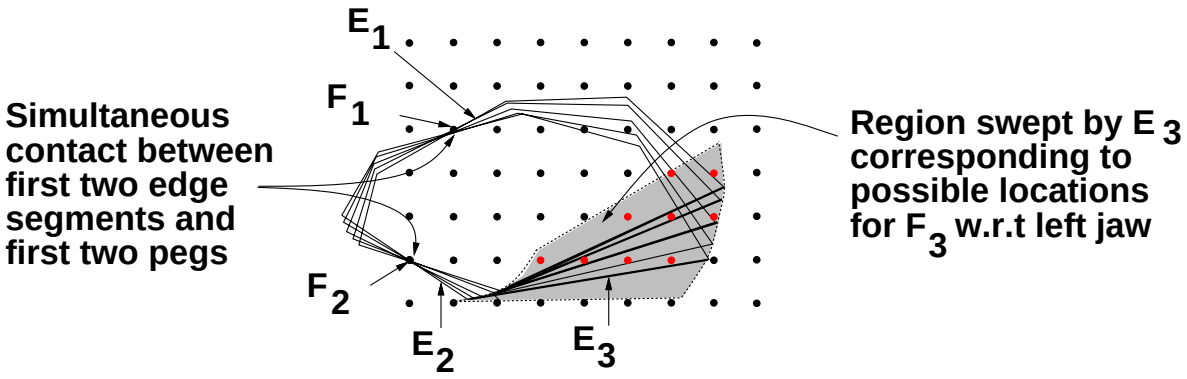


Figure 5.17: The third peg must lie in the region swept by $\mathcal{E}_3$ while maintaining contact between $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{F}_1$, $\mathcal{F}_2$.

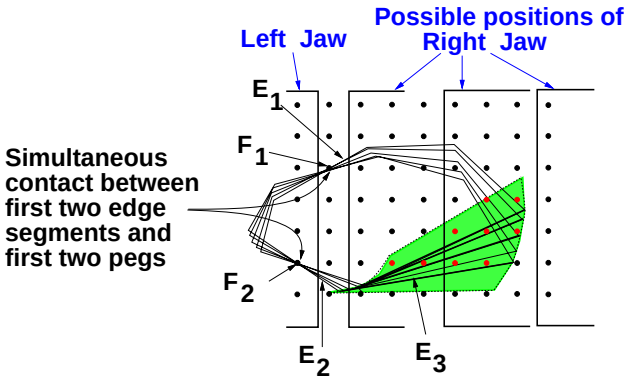


Figure 5.18: In Type I cases, the right jaw is free to translate, so the swept region does not directly specify the set of possible positions for the third and fourth pegs.

Algorithm Listing

Given a quartet of jaw specified edge segments $\vec{\mathcal{E}}$, the code below enumerates all possible quartets of fixture peg positions $\{\vec{\mathcal{F}}, \vec{\mathcal{F}}', \vec{\mathcal{F}}'', \ldots\}$.

1. Assume $\mathcal{F}_1$, the first peg position, is at the left jaw origin.

2. Compute the set of possible positions for the second peg $\{\mathcal{F}_2, \mathcal{F}_2', \mathcal{F}_2'', \ldots\}$:

   (a) Compute ranges of orientations for which at least one point on each left-jaw specified edge lies to the left of at least one point on each right-jaw specified edge (section 5.4.2).

   (b) Enumerate all of the possible positions for the second fixture peg $\{\mathcal{F}_2, \mathcal{F}_2', \mathcal{F}_2'', \ldots\}$ corresponding to lattice points inside sections of an annulus (section 5.4.2).

3. For each second fixture position $\mathcal{F}_2$, reparameterize the object's pose in terms of a single variable, χ, since the first two features must maintain contact with the first two pegs. χ corresponds to the orientation of the orientation of the extended intersection, $P(\chi)$ of $\mathcal{E}_1$, $\mathcal{E}_2$ with respect to the center of a circle C (section 5.4.2). Then, compute the valid ranges of the extended intersection χ: $\{[\chi_{\min}, \chi_{\max}], [\chi'_{\min}, \chi'_{\max}], \ldots\}$

4. For Type I configurations (section 5.4.2):
   Compute the sets of possible positions of the third and fourth pegs:

   (a) Enumerate the rows each edge can cross by computing the minimal and maximal y values of the region swept over by the third and fourth edges (section 5.4.2).

   (b) Compute range of relative x positions between the third and fourth edges along each respective pair of rows (section 5.4.2).

   (c) Enumerate pairs of third and fourth peg positions consistent with the relative range for each respective pair of rows $\{(\mathcal{F}_3, \mathcal{F}_4), (\mathcal{F}_3', \mathcal{F}_4'), (\mathcal{F}_3'', \mathcal{F}_4''), \ldots\}$.

5. For Type II configuration (section 5.4.2):
   Compute the set of possible positions of the third peg:

   (a) Enumerate the rows crossed by the third edge by computing the minimal and maximal y values achieved by the third edge (section 5.4.2).

(b) Enumerate all of the lattice points on each row by computing range of relative x positions of the third edge along each row (section 5.4.2) $\{\mathcal{F}_3, \mathcal{F}_3', \mathcal{F}_3'', \ldots\}$.

(c) Compute the poses which achieve simultaneous contact between the first three edges and the first three fixture elements and for each simultaneous contact pose, transform the fourth edge accordingly, and enumerate one position on each row on the left jaw consistent with the fourth edge $\{\mathcal{F}_4, \mathcal{F}_4', \mathcal{F}_4'', \ldots\}$.

5.4.2 Algorithm Details

In this section, we sketch the subroutines, which are described in more detail in [WC94].

Computing Orientation Ranges Consistent With Left/Right Constraints (2(a))

We constrain the set of possible orientations by exploiting the fact that the contact points (pegs) on the left jaw must lie to the left of the contact points (pegs) on the right jaw; *i.e.*, there must be a point on a left-jaw specified edge which lies to the left of a point on a right-jaw specified edge.

Enumerating Possible Positions of the Second Peg (2(b))

The possible positions of the second peg correspond to lattice points within a section of an annulus. The distance between $\mathcal{F}_1$ and $\mathcal{F}_2$ must agree with the range of distances between points from $\mathcal{E}_1$ and $\mathcal{E}_2$, and the orientation between $\mathcal{F}_1$ to $\mathcal{F}_2$ must be consistent with the Minkowski sum of the ranges satisfying the left/right constraints and the ranges of orientations between points on the edges.

Parameterization for Maintaining Contact Between Two Lines and Two Points (3)

One interesting aspect of this algorithm is that the pose parameterization implicitly maintains contact between the first two edges and fixture pegs. Parameterizing the pose by a single variable simplifies the task of computing the region swept by the third and fourth edges The object's pose is parameterized by the position of the extended intersection of the first two edge segments. The valid extended intersections sweep out a circular arc. This parameterization implicitly satisfies the contact constraints because of the geometric

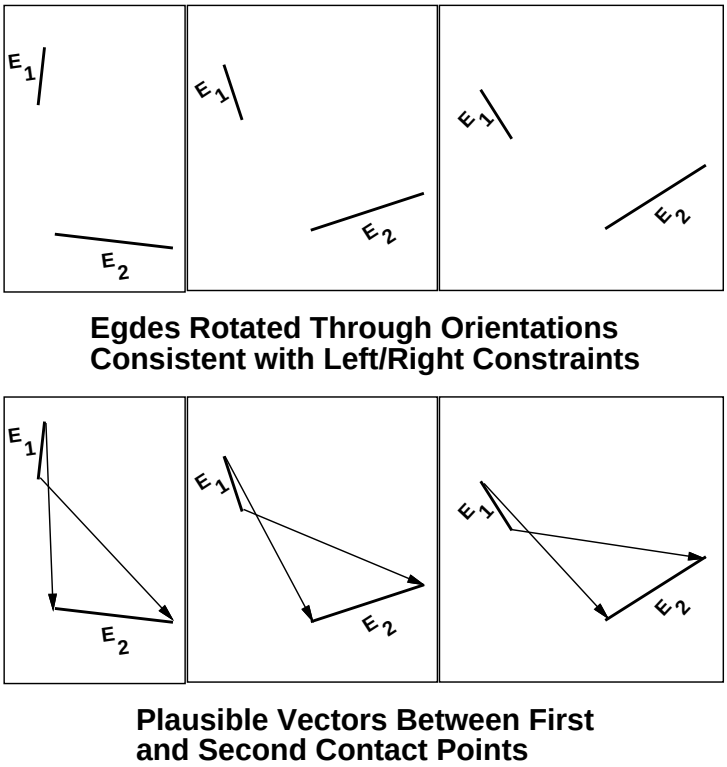


Figure 5.19: The admissible orientations between the first and second contact point depend jointly upon the ranges of orientations satisfying the left/right constraints, and the ranges of orientations of vectors between the two edges

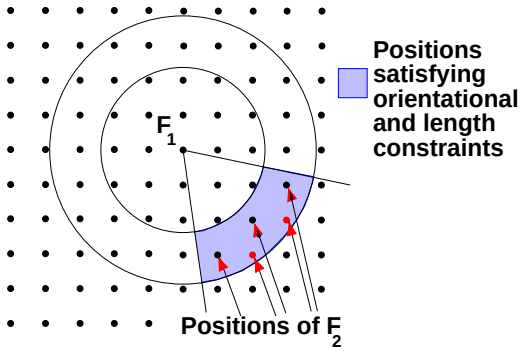


Figure 5.20: The possible positions for the second peg $\{\mathcal{F}_2, \mathcal{F}_2', \mathcal{F}_2'', \ldots\}$ correspond to the lattice points inside the annulus centered at the origin where the minimum and maximum distance constraints are due to the edge segments, and the wedge orientations depend upon both the orientations of the vectors between the two edge segments and the orientations satisfying the left/right constraints.

property that the interior angle between a point on a circle's boundary and a circular arc remains constant.

Let χ be the orientation of the extended intersection ($P(\chi)$) of the two edge segments, C be the circle including the two contact points $\mathcal{F}_1$, $\mathcal{F}_2$, and let A be an arc defined by the two points (Figure 5.21). Contact is maintained because the interior angle of A with respect to $P(\chi)$ is a if and only if $P(\chi)$ is on C's boundary (Figure 5.21). The parameterization breaks down when the two lines are parallel, but those cases are irrelevant (section 5.2.2).

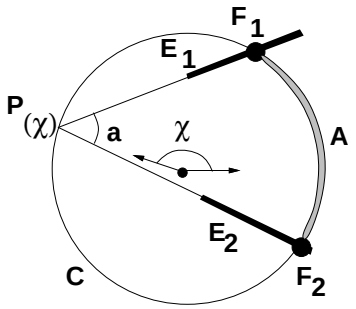


Figure 5.21: Parameterizing the object's pose by the position of the extended intersection χ on the circle's boundary maintains contact between $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{F}_1$, $\mathcal{F}_2$.

The valid χ ranges $\{[\chi_{\min}, \chi_{\max}], [\chi'_{\min}, \chi'_{\max}], \ldots\}$ correspond to three types of constraints: first, χ is constrained to be consistent with the left/right orientational constraints; second, the finite-length edge segments, must, in fact, contact the corresponding peg positions, and thirdly, we also impose additional assumptions that the third or fourth edge segments intersect a particular horizontal lattice row.

Region Swept By Edge $\mathcal{E}$ While Maintaining Contact Between First Two Edges and Pegs (4(a,b),5(a,b))

Next, we compute the region swept over by the third or fourth edge while maintaining contact between the first two edges and the first two pegs. Fortunately, since we are working with a lattice, we do not need to actually compute the region swept over by the third or fourth edge, we only need to compute with the x intersection of the region along the lattice rows.

Computing Lattice Rows Swept Over by Edge $\mathcal{E}$ (4(a),5(a))

The lattice rows crossed by the edge segment $\mathcal{E}$ are computed by determining the continuous range of y coordinates covered by $\mathcal{E}$, while maintaining contact between the first two edge segments and two pegs (Figure 5.22). Determining the lattice rows which intersect the swept region corresponds to determining the minimum and maximum y values of the region (since the lattice rows are of the form $y = k\lambda_y | k \in \mathbb{I}$). Our approach for computing the extremal y values of the region involves computing the range of y values for each endpoint of the edge segment; this because extremal y coordinates must be due to vertices v_1, v_2 of $\mathcal{E}$ (Figure 5.23). The range of y values for each endpoint is computed by formulating $Y(\chi)$, the y position of the endpoint as a function of χ and then checking the extremal χ values $\{\chi_{\min}, \chi_{\max}, \chi'_{\min}, \chi'_{\max}, \ldots\}$, as well as the $\chi*$ values corresponding to local extrema of the map: $\{\chi * | (Y'(\chi*) = 0) \wedge (\chi* \in \{[\chi_{\min}, \chi_{\max}], [\chi'_{\min}, \chi'_{\max}], \ldots\})\}$. Finally, we recompute the valid χ ranges for each lattice row by finding the χ values for which one of the vertices of the edge segment crosses the lattice row, and the χ values corresponding to these cases are found by solving $Y(\chi') = k\lambda_y$.

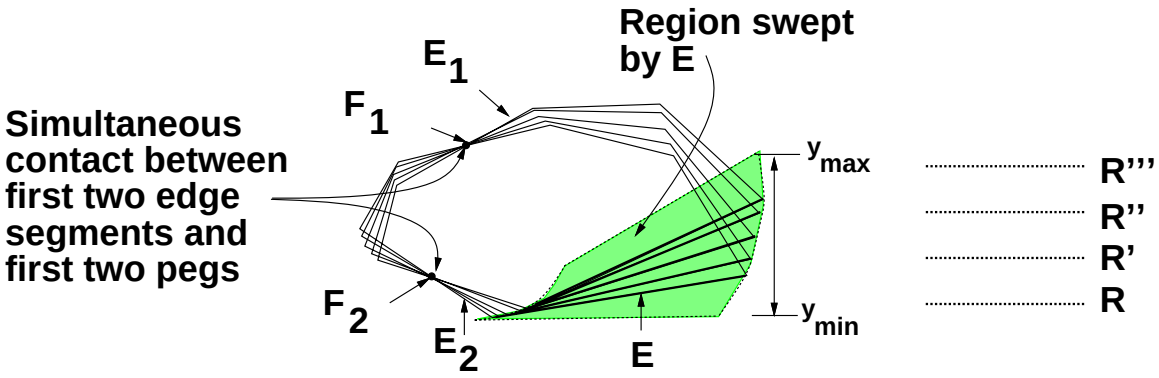


Figure 5.22: The lattice rows intersecting the region swept by $\mathcal{E}$ while maintaining contact between $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{F}_1$, $\mathcal{F}_2$.

Computing X Coordinate Range of $\mathcal{E}$ Along Lattice Row $\mathcal{R}$ (4(b),5(b))

In this section we describe the computation of $[X_{\mathcal{E}}^{\mathcal{R}}]$, the range of x coordinates (with respect to the left jaw origin) along a row $\mathcal{R}$ ($y = k\lambda_y$) swept over by edge $\mathcal{E}$ (Figure 5.24) while maintaining contact between the first two edges and two pegs. Again, we compute the minimum and maximum values of the range $x_{\min}, x_{\max}$ by formulating an alge-

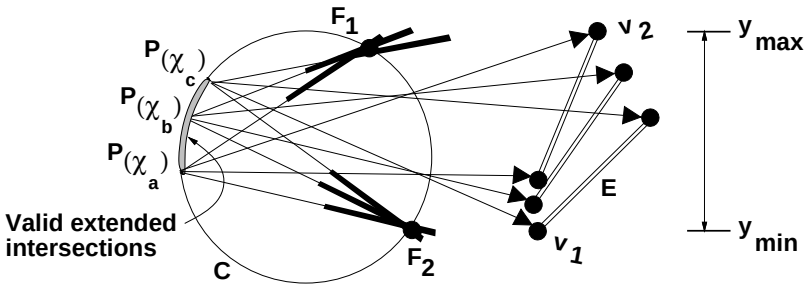


Figure 5.23: The y range of the region swept by edge $\mathcal{E}$ is found by computing the y ranges of the $\mathcal{E}$'s vertices v_1, v_2 while maintaining contact between $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{F}_1$, $\mathcal{F}_2$.

braic expression. In this case, we formulate the x coordinate the extended intersection $X(\chi)$ between the lattice row $\mathcal{R}$ and the edge $\mathcal{E}$ as a function of χ. We check the extremal values of χ $\{\chi_{\min}, \chi_{\max}, \chi'_{\min}, \chi'_{\max}, \ldots\}$, as well as the χ values corresponding to local extrema of the map: $\{\chi * | (X(\chi*) = 0) \wedge (\chi* \in \{[\chi_{\min}, \chi_{\max}], [\chi'_{\min}, \chi'_{\max}], \ldots\})\}$.

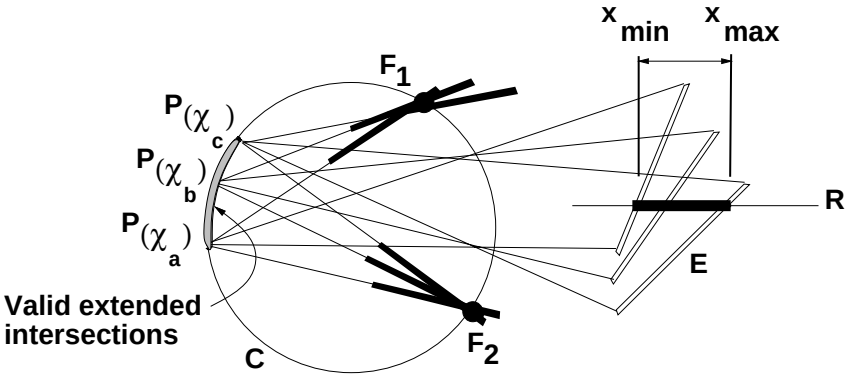


Figure 5.24: We compute the range $[X^{\mathcal{R}}_{\mathcal{E}}]$ along lattice row $\mathcal{R}$ swept over by $\mathcal{E}$ while maintaining contact between $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{F}_1$, $\mathcal{F}_2$.

Enumerating Peg Positions $\mathcal{F}_3$, $\mathcal{F}_4$ for Type I Configurations (4(c))

Enumerating the third and fourth peg positions involves enumerating lattice points corresponding to regions defined in terms of a different coordinate frame. Since we performed all of our computations with respect to the origin on the left jaw, we computed the regions swept by the third and fourth edges with respect to the left jaw. At this point, these regions do not specify the possible peg positions since the third and fourth pegs reside on the right jaw; therefore, we need to compute the *difference* between the ranges of x

positions of the swept regions to enumerate lattice points. We follow the convention that the x coordinate of the third peg is equal to zero ($\mathcal{F}_3.x = 0$).

Consider all pairs of rows corresponding to the two edge segments: $(\mathcal{R}, \mathcal{R}^*)$. Even though the x ranges $[X^{\mathcal{R}}_{\mathcal{E}_3}]$, $[X^{\mathcal{R}^*}_{\mathcal{E}_4}]$ were computed with respect to the left jaw, the differential between the x coordinate ranges $[X^{\mathcal{R}}_{\mathcal{E}_3}] \oplus (\Leftrightarrow[X^{\mathcal{R}^*}_{\mathcal{E}_4}])$ for $\mathcal{E}_3$ and $\mathcal{E}_4$ is invariant with respect to translation in x. Since both edges are expected to lie on the left jaw, we intersect both ranges with $[\max(\mathcal{F}_1.x, \mathcal{F}_2.x), \infty]$. We enumerate a pair of fixture peg positions $(\mathcal{F}_3, \mathcal{F}_4)$ for each discrete value $(k\lambda_x \in [X^{\mathcal{R}}_{\mathcal{E}_3}] \oplus (\Leftrightarrow[X^{\mathcal{R}^*}_{\mathcal{E}_4}])(k \in \mathbb{I}))$ in the range (Figure 5.25).

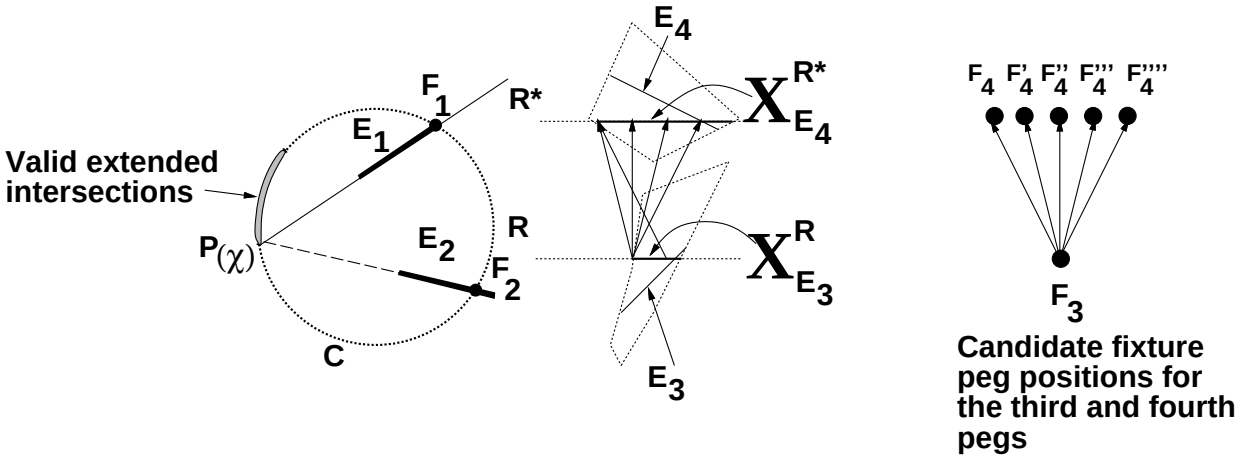


Figure 5.25: We enumerate all of the discrete vectors corresponding to plausible fixture positions $\{(\mathcal{F}_3, \mathcal{F}_4), (\mathcal{F}'_3, \mathcal{F}'_4), \ldots\}$ using the x coordinate ranges swept over by the third and fourth edges $X^{\mathcal{R}}_{\mathcal{E}_3}, X^{\mathcal{R}}_{\mathcal{E}_4}$.

Enumerating Peg Positions $\mathcal{F}_3$, $\mathcal{F}_4$ for Type II Configurations (5(c))

For Type II situations, since the third peg also resides on the left jaw, we simply enumerate all of the lattice points within the regions $k\lambda_x \in [X^{\mathcal{R}}_{\mathcal{E}_3}]$, $(k \in \mathbb{I})$ along rows $\{\mathcal{R}, \mathcal{R}', \ldots\}$. Then, we compute the object poses corresponding to simultaneous contact between the first three edge segments and the first three pegs. Then, we enumerate all of the lattice rows which overlap the fourth edge, and enumerate a fourth peg position with $\mathcal{F}_4.x = 0$ for each of those rows.

Since we are already know the object's pose, we can efficiently determine the set of possible contact points on the fourth edge segment which achieve force closure. Given the object's pose and three contacts and the edge of the remaining contact, we know the

forces and torques corresponding to three of the contacts, and the direction of the force of the fourth contact; the only unknown is the torque τ_4. In order to achieve force closure, all of the determinants of the minors of α_i must be of the same sign. Three of the minors are linear functions of τ_4, and the fourth is constant. We constrain the linear functions to be of the same sign as the constant minor, arriving at a range of torques necessarily and sufficiently producing force closure. From the range of torques, we can compute the range of contact positions along the fourth edge. Pruning the fourth peg position in this manner will improve the algorithm's performance for Type II contacts by only enumerating force closure configurations. Unfortunately, we know of no similar heuristic for Type I cases because the object's pose cannot be determined given only three contacts because there are an infinite number of possible poses since the third peg can still translate in x with respect to the first two contacts.

5.4.3 Algorithmic Complexity

In this section, we derive an upper bound on the algorithm's complexity by presenting an upper bound on A, the number of simultaneous contact configurations. Our worst case analysis involves computing the number of simultaneous contact configurations for a canonical edge quartet, and multiplying this amount by $O(n^4)$ to account for all edge quartets.

The canonical edge quartet is characterized by three values δ, ρ, Δ which are functions of the object O. δ refers to the maximum distance between any two points in O, ρ refers to the largest range of distance between points on two edge segments, and Δ refers to the length of the longest edge ($\delta \geq \rho$, $\rho > \Delta$, and often $\delta >> \rho$).

First, we compute the bound for Type II contacts, where the second and third pegs must lie within an annulus of area $2\pi(2\delta\rho + \rho^2)$, centered at the origin (the position of the first peg); this yields $O((\frac{\delta\rho}{\lambda_x\lambda_y})^2)$ possible positions for the first three contacts. Since the object's pose(s) can be determined given the first three contacts, the fourth peg must lie along a row covered by the fourth edge (assuming that the fourth peg lies in the leftmost column), yielding $O(\frac{\Delta}{\lambda_y})$ possible peg positions. Therefore, there are at most $O((\frac{\delta\rho}{\lambda_x\lambda_y})^2\frac{\Delta}{\lambda_y})$ Type II contacts for any quartet of edges.

For Type I contacts, the third and fourth peg positions reside on the right jaw, and are therefore free to translate, which implies that we cannot simply count lattice positions

inside the annulus regions. Again, there are $O(\frac{\delta\rho}{\lambda_x\lambda_y})$ possible second peg positions. We count the number of possible third and fourth peg positions by counting the number of relative (with respect to x translation) discrete lattice positions between the third and fourth regions. The annulus crosses at most $O(\frac{\delta}{\lambda_y})$ rows, covering $O(\sqrt{\delta\rho})$ x along each row; this yields $O((\frac{\delta}{\lambda_y})^2\frac{\delta}{\lambda_x})$ possible positions for the third and fourth pegs, producing an overall bound of $O(\frac{\delta^3\rho}{\lambda_x^2\lambda_y^3})$ for the total number of Type I configurations for any quartet of edges.

Therefore, the number of simultaneous contact configurations, and consequently, the running time, A, is bounded by configurations of the first type $O(n^4(\frac{\delta^3\rho}{\lambda_x^2\lambda_y^3}))$. Note that if $\lambda_y = \lambda_x = 1$ and $\delta = \rho$ then the complexity is $O(n^4\delta^5)$, where δ is the object's diameter expressed in terms of lattice units. This estimate agrees with the number of configurations found in practice; for example, Table 5.2 shows that for the Hexnut (section 5.6.2) object with spacing $\lambda_y = \lambda_x = 3.0$, there were 1495 configurations, and with spacing $\lambda_y = \lambda_x = 2.0$, there were 12643 configurations ($(\frac{3}{2})^5 = 7.59375 \approx \frac{12643}{1495} = 8.45686$).

5.5 Computing Simultaneous Contact Poses

This section describes a technique for computing the object's poses achieving simultaneous contact between the edge segments $\vec{\mathcal{E}}$ and pegs $\vec{\mathcal{F}}$ (Figure 5.26). To compute the poses of simultaneous contact, the constraints corresponding to the relative displacements between contacts on the same fixture jaw are formulated algebraically. Each pair of contacts on the same jaw provides one such constraint, and we compute the object's pose by intersecting two such constraints.

Since we are trying satisfy x difference constraints between two contacts, the poses are parameterized in terms of θ and y. We formulate the difference in x between two contacts as a function of θ and y where (θ, y) represents the object after a rigid two-dimensional transformation of rotation by θ and translation by y parallel to the y-axis; x is assumed to be zero since both jaws can translate freely along the x axis.

In both Type I and Type II situations, there are two independent relative displacements between contact points: in Type I situations, each jaw's pair of contacts supplies a relative displacement constraint, and in Type II situations, any two pairwise contact combinations from the left jaw supply two relative displacement constraints. For each quartet of edges and pegs, there are at most four distinct Type I poses and at most two distinct

Type II poses.

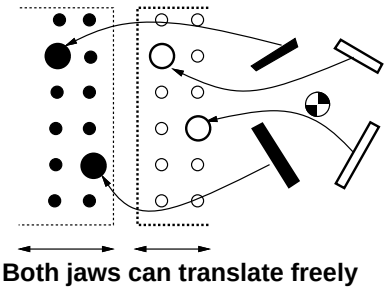


Figure 5.26: Determine the pose such that edge segments $\vec{\mathcal{E}}$ simultaneously contact pegs $\vec{\mathcal{F}}$ on fixture jaws which freely translate along the x axis.

The poses achieving simultaneous contact are computed algebraically. First we formulate extended intersection functions (section 5.5.1) expressing the x coordinate of the extended intersection between a horizontal line and an edge segment rotated around the origin by θ, and translated in y. Next, we formulate extended intersection difference functions which correspond to the difference between two extended intersection functions, and then we define curves corresponding to achieving a desired relative x position. Finally, we compute the object poses satisfying both pairs of relative contact constraints by intersecting the curves.

We compute the intersections of such curves algebraically by setting up a system of multivariate expressions, and solving for the simultaneous zeros of these expressions. Our approach involves utilizing a trigonometric substitution $t = \tan(\frac{\theta}{2})$ to arrive at algebraic expressions (section 3.5.5).

5.5.1 Extended Intersection Functions $I(\theta, y)$

Extended intersection functions, $I(\theta, y)$ algebraically describe the x coordinate of the extended intersection between a horizontal lattice row and a corresponding edge segment $\mathcal{E}_\alpha$ rotated around the reference point by θ and translated by y (Figure 5.27). Extended intersection functions $I(\theta, y)$ are related to $X^{\mathcal{R}}_{\mathcal{E}}$, the range of x coordinates swept by an edge along a row; the major distinction between these abstractions is that the extended intersections $I(\theta, y)$ are parameterized by θ and y, the object's pose, whereas the row contacts $X^{\mathcal{R}}_{\mathcal{E}}$ were parameterized by χ, the orientation of the extended intersection.

$I_\alpha(\theta, y)$ is defined in equation (5.3) in terms of θ, y, R_α, D_α, α. α is the orientation

normal to $\mathcal{E}_\alpha$ pointing outward. R_α is the minimum distance from the reference point to $\mathcal{E}_\alpha$, and D_α is the difference in y coordinates between the modular row $\mathcal{R}$ and the reference point.

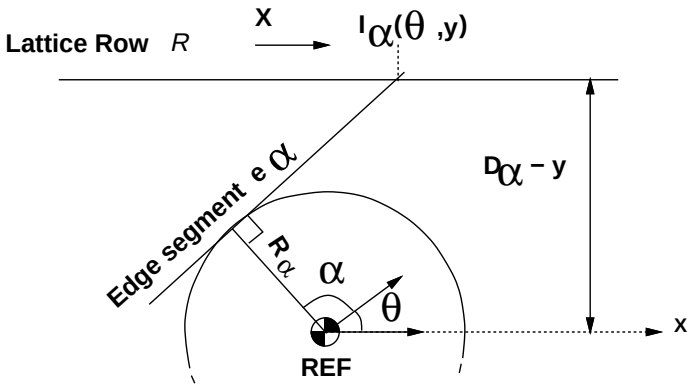


Figure 5.27: The extended intersection $I(\theta, y)$ between edge segment $\mathcal{E}_\alpha$ and lattice row $\mathcal{R}$.

$$I_\alpha(\theta, y) = R_\alpha \cos(\theta + \alpha) \Leftrightarrow \tan(\theta + \alpha)(D_\alpha \Leftrightarrow y \Leftrightarrow R_\alpha \sin(\theta + \alpha)) \tag{5.3}$$

5.6 Results

In this section, we present example fixture vise configurations generated by our algorithm. We also present results on the number of valid fixture designs for various objects. The fixture counts already discount object and lattice symmetries by only counting one fixture for each family of translated or symmetric fixtures.

5.6.1 Hexagonal Object

In order to understand the number of configurations for different numbers of available peg radii, the fixture design algorithm was run for the hexagonal object shown in Figure 5.28. The first run utilized only pegs of radius 2.0, and the second run used pegs of radii 1.5 and 2.0. Two fixture designs from the first run are shown in Figure 5.29 and two fixture designs from the second run, in which multiple radii pegs were used, are shown in Figure 5.30. Table 5.1 presents the number of valid fixture vise configurations for various λ_x, λ_y spacings and peg radii sets.

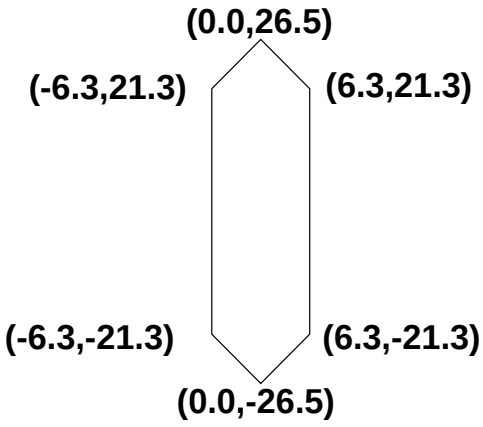
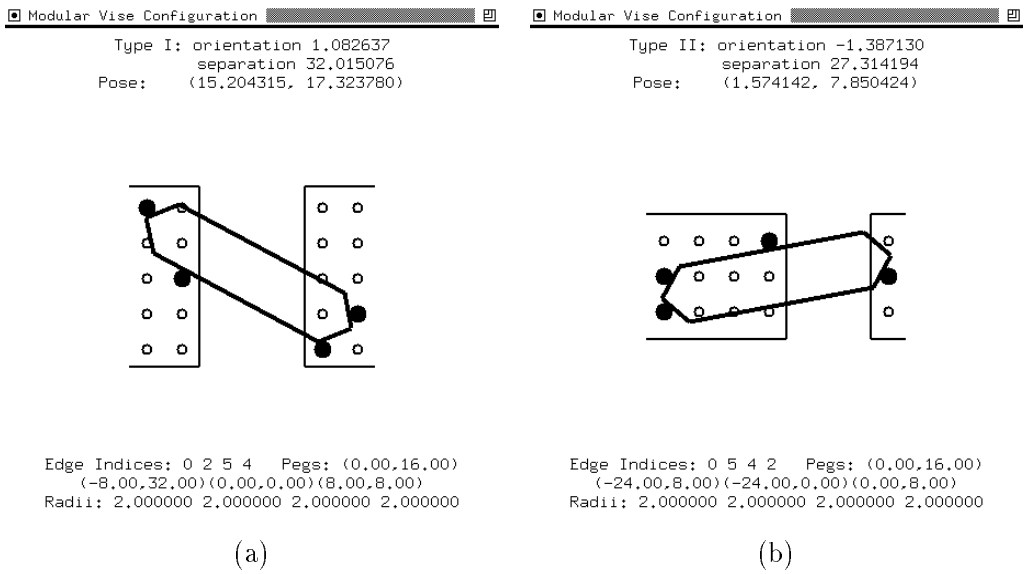


Figure 5.28: The fixture design algorithm was run for the hexagonal object shown above

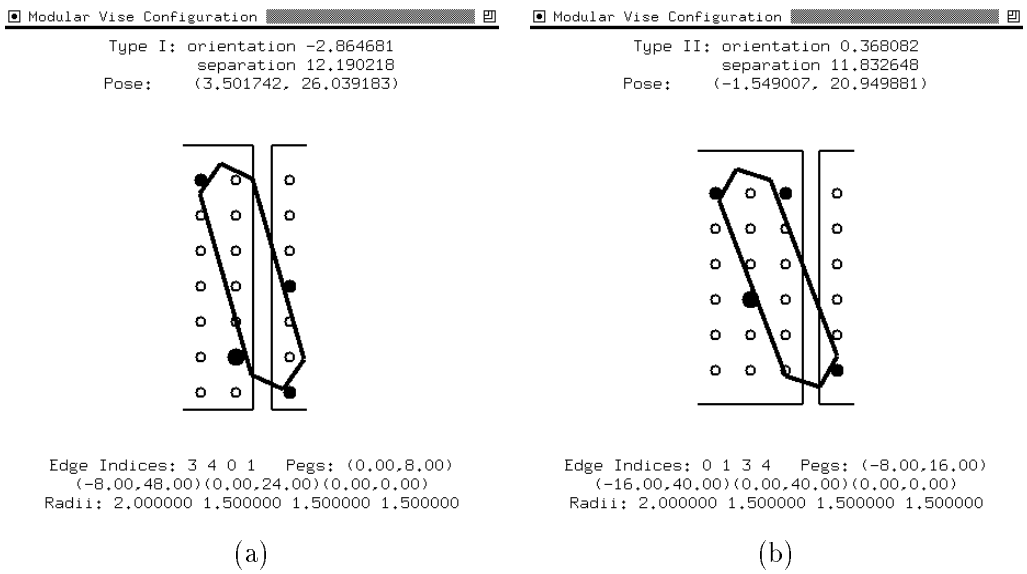


(a) (b)

Figure 5.29: (a): Type I workholding for a hexagonal object. (b) Type II workholding for a hexagonal object.

Object	Spacing (λ_x, λ_y)	Peg Sizes	Total # Configurations	# Type I	# Type II
Hexthing	8.0, 8.0	2.0	726	337	389
Hexthing	8.0, 8.0	1.5, 2.0	8950	3967	4983
Hexthing	6.0, 6.0	2.0	2962	1190	1772
Hexthing	6.0, 6.0	1.5, 2.0	36835	14346	22489

Table 5.1: The number of fixture design configurations for a hexagonal object with varying sets of pegs.



(a) (b)

Figure 5.30: (a) Type I workholding for a hexagonal object using different radii pegs. (b) Type II workholding for a hexagonal object using different radii pegs.

5.6.2 Hexnut

The fixture design algorithm was also run for the hexnut object shown in Figure 5.31. The first run utilized only pegs of radius 1.0, and the second run used pegs of radius 0.75 as well as pegs of radius 1.0. Fixture designs are shown in Figure 5.32. Table 5.2 presents the number of valid fixture vise configurations for various λ_x, λ_y spacings and peg radii sets.

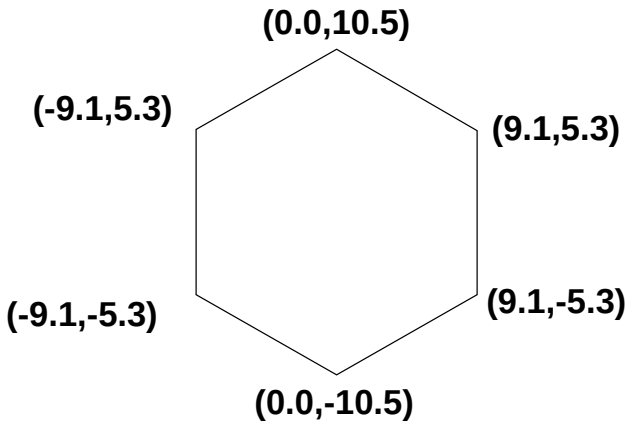
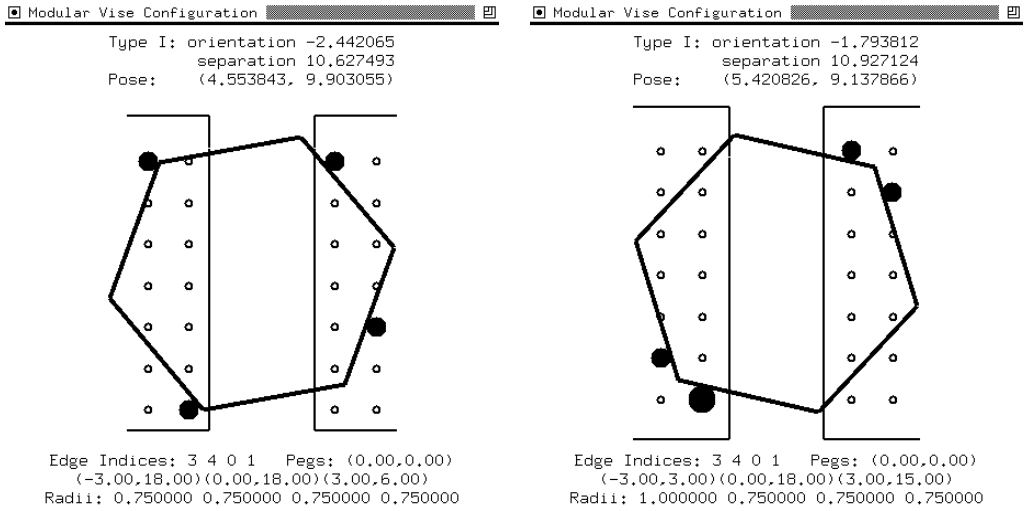


Figure 5.31: The fixture design algorithm was run for the hex nut shown above

Object	Spacing (λ_x, λ_y)	Peg Sizes	Total # Configurations	# Type I	# Type II
Hexnut	3.0, 3.0	1.0	1495	639	856
Hexnut	3.0, 3.0	0.75, 1.0	20912	8320	12592
Hexnut	2.5, 2.5	1.0	3819	1825	1994
Hexnut	2.0, 2.0	1.0	12643	5391	7252

Table 5.2: The number of fixture design configurations for a hexnut with varying sets of pegs.



(a) (b)

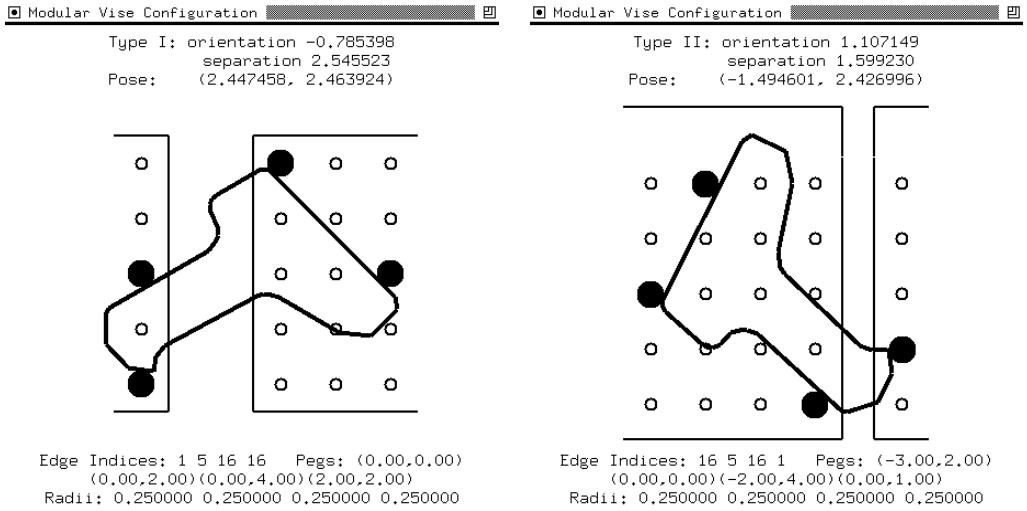
Figure 5.32: (a) Type I workholding for a hexnut object using same radii pegs. (b) Type I workholding for a hexnut object using different radii pegs.

5.6.3 Gluegun

The fixture design algorithm was also run for the gluegun object examined by Brost and Goldberg [BG94]. The first run utilized only pegs of radius 0.25, and the second run utilized pegs of radius 0.5. Fixture designs are shown in Figure 5.33. Table 5.3 presents the number of valid fixture vise configurations for various λ_x, λ_y spacings.

Object	Spacing (λ_x, λ_y)	Peg Sizes	Total # Configurations	# Type I	# Type II
Gluegun	0.75, 0.75	0.25	3685	2473	1212
Gluegun	1.0,1.0	0.25	779	538	241

Table 5.3: The number of fixture design configurations for the gluegun with varying sets of pegs.



(a) (b)

Figure 5.33: (a) Type I workholding for a glue gun. (b) Type II workholding for a glue gun.

5.7 Conclusion

In this chapter, we described a complete, efficient algorithm for designing fixture vise configurations for prismatic objects. This algorithm can be an integral part of an interactive fixture design system. A complete enumeration of fixture configurations is a prerequisite for a multi-step fixture design planner. The two key subroutines involve enumerating all fixture element configurations possibly achieving simultaneous contacting

between four edge segment and four pegs, and computing the object's pose and vise separation achieving simultaneous contact between a particular quartet of edges and a particular quartet of fixture elements.

Chapter 6

Fixture Design for Generalized Polyhedra

Abstract

The term fixturing refers to the task of immobilizing a workpiece for the purposes of performing assembly and machining operations. Fixturing is a fundamental problem in manufacturing. Fixtures can either be fabricated from scratch, or assembled from a toolkit of modular components, the latter approach is termed modular fixturing. In Chapter 6, we presented a complete fixture design algorithm [WC94] which automatically designs fixtures for prismatic polyhedra using a fixture vise toolkit. The algorithm enumerates configurations of pegs inserted into lattices of holes, and the lattices are mounted on jaws of a vise. In this chapter, we extend the algorithm to handle generalized polyhedral prismatic workpieces; a generalized polyhedral prismatic workpiece has a generalized polygonal projection on the fixture plane, where a generalized polygon refers to a planar object having a boundary composed of linear and circular arcs.

6.1 Introduction

The task of immobilizing a workpiece via mechanical devices, commonly called fixturing or workholding, is an essential task in manufacturing. Fixtures must be able to immobilize objects while resisting forces on the order of 20KN for machining, and 50N for

assembly. Fabricating fixtures from scratch is expensive and time-consuming, compared to assembling fixtures from a modular toolkit of components; the latter approach is termed modular fixturing, and provides cost effectiveness, high precision, strength, speed, and reconfigurability.

In the past, fixture design has been more of a craft than a science. Although automated fixturing systems have been developed, most such systems rely on knowledge engineering to imitate an expert machinist's analysis. Expert systems have the drawback that without fundamental geometric analysis, they are only capable of describing types of fixtures, they cannot compute optimal fixture configurations and corresponding workpiece poses.

Recent results have sparked renewed interest in modular fixture design; two related algorithms have been proposed and implemented: Wallack and Canny's fixture vise algorithm [WC94] (Chapter 6) and Brost and Goldberg's translating clamp algorithm [BG94]. Both algorithms are efficient and completely enumerate configurations capable of immobilizing a given prismatic polyhedral workpiece. Both these approaches are successful because they focus on a solvable problem with a finite number of solutions, such that simple brute force generate and test strategies are sufficient.

In Chapter 6, Wallack and Canny presented a fixture vise toolkit, which is based on a device similar to the Black and Decker Workmate. For the fixture vise, modular fixture elements (pegs) are inserted into lattices of holes (fixture tables), and the fixture tables are mounted on jaws of a vise (Figure 6.1). The fixture vise can alternatively be viewed as an universal gripper.

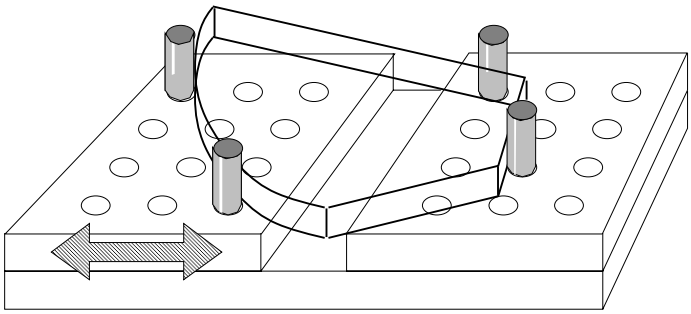


Figure 6.1: A fixture vise consists of two fixture table jaws capable of translating in x.

In this chapter, we extend the fixture vise algorithm to handle generalized polyhe-

dral workpieces; *i.e.*, we present a complete polynomial time algorithm for computing fixture configurations for immobilizing prismatic generalized polyhedral workpieces (objects with generalized polygonal projections). Our approach focuses on the planar analogue to the three dimensional problem (Figure 6.2). By focusing on the two dimensional problem, we can ignore the out of plane forces and torques; outside the scope of this work, other devices will will be utilized to handle these forces and torques.

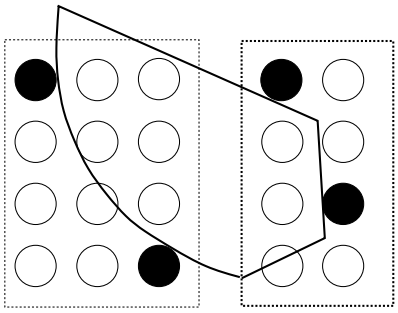


Figure 6.2: The two-dimensional view of the workpiece and the fixture vise in Figure 1.

6.1.1 Related Work

This work is related to two recently developed modular fixturing algorithms, the fixture vise design algorithm developed by Wallack and Canny [WC94] (Chapter 5) and the translating clamp toolkit design algorithm developed by Brost and Goldberg [BG94]. These algorithms are closely related in that both algorithms were implemented for planar polygonal models, both involve generate and test techniques, and both focus on systems with a single degree of freedom. Wallack and Canny's fixture vise allows for two different types of configurations: Type I- where two pegs reside on each jaw of the vise, and Type II- where three pegs reside on one jaw of the vise, and one peg resides on the other. Brost and Goldberg's system involves inserting three pegs and a translating clamp into a fixture table lattice. It turns out that the translating clamp toolkit usually provides fewer fixtures than the fixture vise toolkit because translating clamp fixture configurations are analogous to Type II fixture vise configurations, where the peg on the right jaw acts as the translating clamp.

There are two main differences between the two algorithms: Wallack and Canny used a more analytical approach to compute a tighter bound on the set of possible peg posi-

tions (Chapter 5), and Brost and Goldberg exploited the fact that for the translating clamp toolkit, the workpiece's pose is determinable after choosing the first three contacts; the latter difference significantly improves performance because Brost and Goldberg enumerated triplets of model features, rather than quartets of model features.

6.1.2 Overview

Our original fixture vise design algorithm was basically a generate and test algorithm. It enumerated quartets of features (model edges), enumerated peg positions for each quartet of features, computed the poses achieving simultaneous contact between the pegs and the features, and verified force closure for simultaneous contact configuration. Extending this algorithm to handle generalized polygonal models involves extending the peg enumeration routine and the simultaneous contact pose computation routine. First, we will describe an extension for the peg enumeration routine, and then we will describe how we extend the simultaneous contact pose computation.

The peg enumeration routines for generalized polygonal models are much less complex than the peg enumeration routines for polygonal models used in the original algorithm. In this algorithm, we use a less complex approach and achieve a looser bound of possible peg configurations. The original, more analytical, technique, was tailored to polygonal models; in the original algorithm, the pose was reparameterized in terms of a single variable after the first two peg positions were chosen (section 5.4.2), and this parameterization was used to compute the region swept over by the other edges. The parameterization in the original algorithm exploited the fact that the extended intersection of the first two linear features sweeps out a circular arc while contact is maintained between the first two features and the first two pegs. Unfortunately, this property is invalid for circular arc features, and this is why we rely on a simpler method to conservatively estimate the regions swept out by the other edges. This method involves wedges of annuli; wedges of annuli are defined in terms of the minimum and maximum distances between points on two features, and the ranges of orientations consistent with vectors connecting the two features.

Of course, we could have computed the peg contact regions by using multiline segments which bounded the circular features. This is a reasonable approach because the regions swept out by a circular feature is necessarily bounded by the regions swept out by bounding straightline models. Still, this approach does not provide for a simple one degree

of freedom parameterization of the object poses after the first two peg contacts are chosen.

The second key distinction between the extended algorithm and the original algorithm is that we compute the simultaneous contact poses in a different manner. Simultaneous poses between circular and linear features and point sets are computed using a combination of classical algebraic methods and numerical techniques. The contact constraints are formalized algebraically, and we use elimination techniques (resultants) (section 3.5.5) to reduce the system to a univariate polynomial so that we can solve for one variable at a time [Mac02; Man92]. In the original algorithm, we computed the simultaneous contact poses in terms of the configuration variables (y, θ); for linear features, computing the simultaneous contact pose involved solving at most a quartic polynomial expression. Extending this naive approach to generalized polygons with circular arc features, would have resulted in 32^{nd} order polynomial expressions, which is disadvantageous because the running times and numerical inaccuracy increase cubically with the degree of the polynomial. Instead, we utilized a parameterization which implicitly satisfied one of the four contact constraints; this approach simplified the task of computing the simultaneous contact poses to solving at worst a 24^{th} order polynomial.

6.1.3 Outline

In section two, we discuss the theoretical background. We present an overview of the algorithm in section three, and in section four, we describe the method for computing the orientation ranges between linear edges and circular edges. In section five, we explain how we compute the workpiece pose and jaw separation for type I and type II configurations. We discuss the performance of these algorithms in section six, and we conclude by highlighting the results and advantages of our technique.

6.2 Theoretical Background

In this section, we describe the notation and observations.

6.2.1 Notation

- The operator $\oplus$ refers to the Minkowski sum which can be defined as: $A \oplus B = \{a + b | a \in A, b \in B\}$; we use the Minkowski sum to map from $S^1 \times S^1 \rightarrow S^1$ and

$\mathbb{R}^1 \times \mathbb{R}^1 \rightarrow \mathbb{R}^1$

- O refers to a prismatic generalized polyhedral workpiece.
- $\vec{\mathcal{E}}$ refers to a quartet of *jaw-specified* edge segments.
- $\vec{\mathcal{F}}$ refers a quartet of peg positions contacting the edge segments $\vec{\mathcal{E}}$.
- λ_y and λ_x refer to the spacing between the rows and columns respectively on the modular jaws (Figure 6.3).

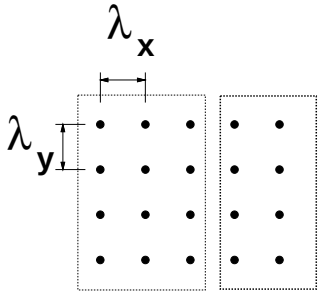


Figure 6.3: λ_y and λ_x refer to the spacing between the rows and columns respectively on the modular jaws.

6.2.2 Observations

To simplify the analysis, we reduce the problem of achieving simultaneous contact with pegs to the problem of achieving simultaneous contact with points by suitably shifting the corresponding edge by the peg radius. Cylindrical or flatted pegs can conservatively be considered point contacts (pegs of zero radius) by transforming the linear or circular features (Figure 6.4). For circular pegs, this reduction is equivalent, but for flatted pegs contacting linear features, this reduction is conservative. This point contact approximation provides a consistent framework for treating both types of contacts. The drawback of this assumption is that we require four pegs per configuration, even though three flatted pegs could achieve force closure.

6.3 Algorithm

In this section, we explain the generate and test subroutines which are sketched in Figures 6.5-6.7.

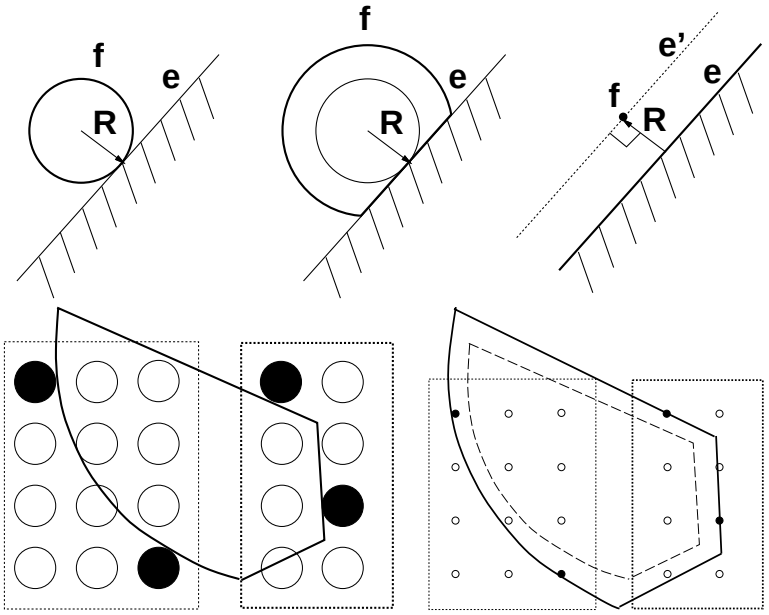


Figure 6.4: Cylindrical or flatted pegs can be considered point contacts by suitably shifting the corresponding edge by the peg radius.

1. **Generate:**

   (a) Enumerate all quartets of jaw-specified edge segments $\{\vec{\mathcal{E}}, \vec{\mathcal{E}}', \vec{\mathcal{E}}'', \ldots\}$ for which we can possibly achieve force closure with four point contacts (Figure 6.6).

   (b) For each edge segment quartet $\vec{\mathcal{E}}$, compute fixture configurations $\{\vec{\mathcal{F}}, \vec{\mathcal{F}}', \vec{\mathcal{F}}'', \ldots\}$ providing simultaneous contact with $\vec{\mathcal{E}}$ (Figure 6.6). The fixture vise configurations $\{\vec{\mathcal{F}}, \vec{\mathcal{F}}', \vec{\mathcal{F}}'', \ldots\}$ are enumerated as shown in Figure 6.7. We assume the first peg resides at the origin, and then we enumerate all of the positions for the second peg. For each set of positions of the first and second pegs, we enumerate all of the positions of the third peg, and for each set of positions of the first, second and third pegs, we enumerate all of the positions of the fourth peg. The possible positions are geometrically constrained by relationships between the corresponding features.

2. **Test:**

   (a) Compute the workpiece poses and jaw separations achieving simultaneous contact between edge segments $\vec{\mathcal{E}}$ and peg positions $\vec{\mathcal{F}}$.

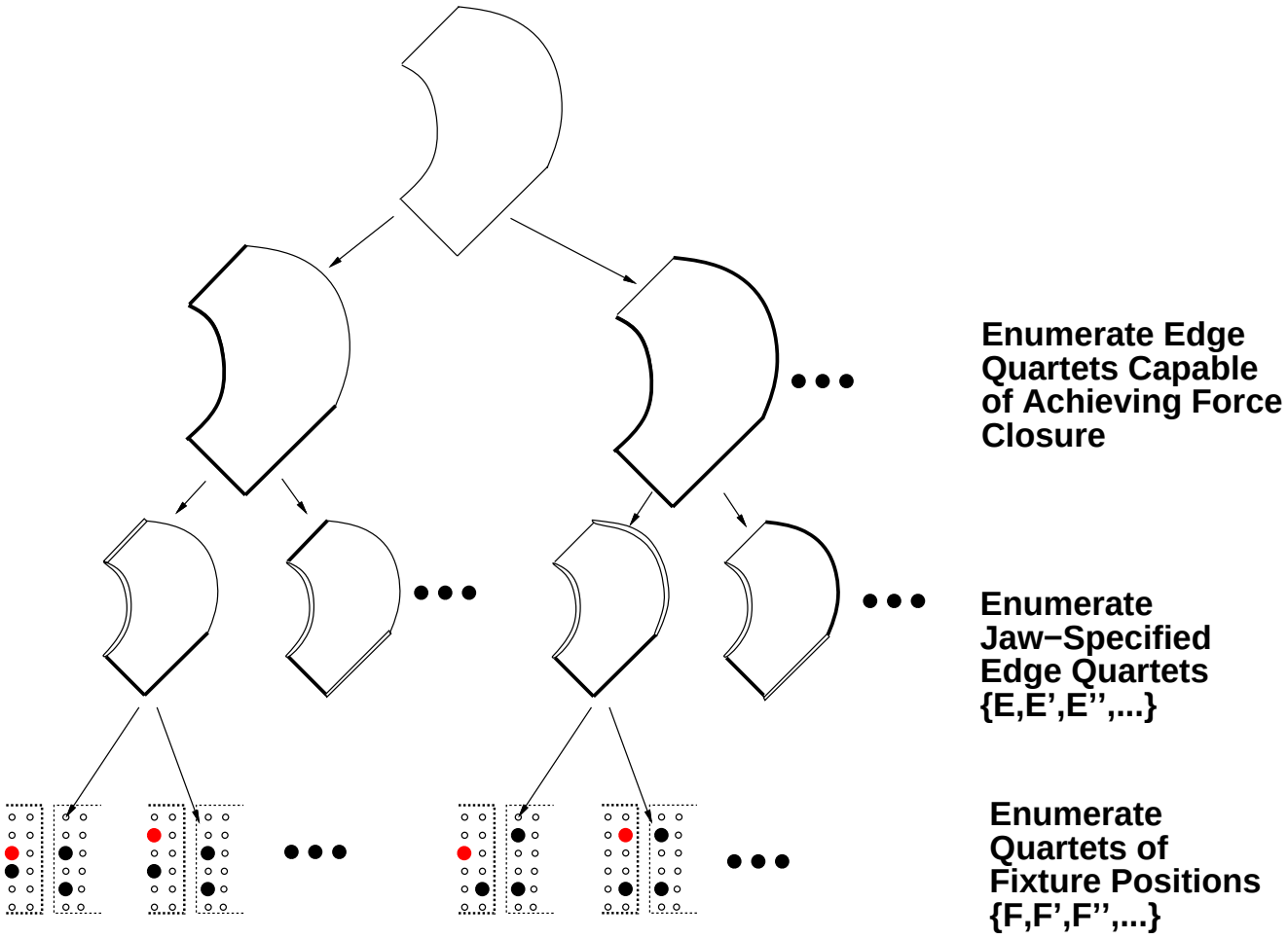


Figure 6.5: The generate portion of the algorithm enumerates quartets of jaw specified edge segments, and for each edge quartet, enumerates quartets of fixture positions (represented by the filled-in circles) capable of achieving force closure.

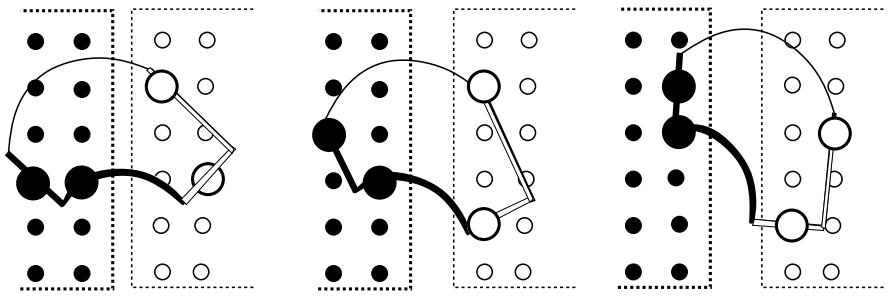


Figure 6.6: Different peg configurations $\vec{\mathcal{F}}$, $\vec{\mathcal{F}}'$, $\vec{\mathcal{F}}''$ simultaneously contacting edge segments $\vec{\mathcal{E}}$.

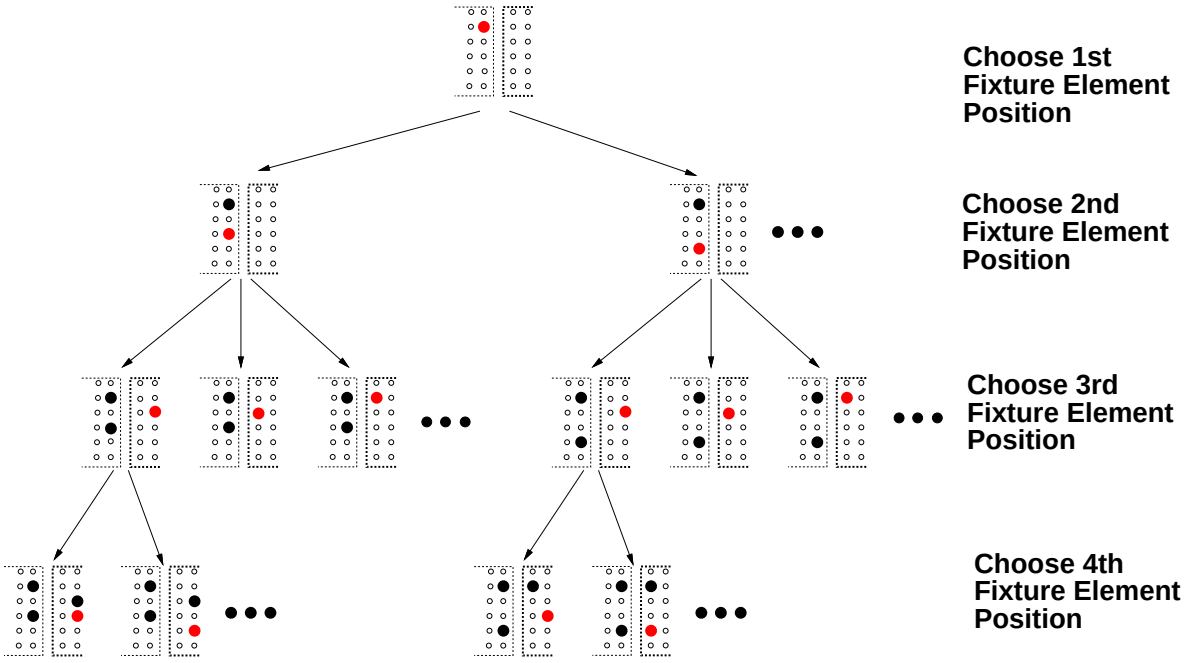


Figure 6.7: The peg configurations are enumerated by picking the position of the second peg, then picking the position of the third peg, etc. (filled-in circles represent the fixel positions).

(b) Verify force closure for each simultaneous contact pose.

6.4 Enumerating Peg Positions

In this section, we describe the technique for enumerating the possible peg positions. This technique uses geometrical constraints to bound the possible positions of each peg corresponding to the given quartet of edges.

The first peg is placed at the origin on the left jaw. The second, third, and fourth peg positions are bounded by geometrical constraints. Since the first peg contacts the first feature and the second peg contacts the second feature, the distance between the two pegs must be consistent with the set of possible distances between points on two features; furthermore, the orientation of the vector from the first peg to the second peg must also be consistent with orientations of vectors between points on the features. The positions of the third and fourth pegs are also bounded by geometrical constraints associated with the respective model features.

We can exploit the lattice structure of the fixture table so that we do not need to completely characterize the regions swept by the second, third, and fourth edge segments. As a consequence of the lattice structure, we either only need to enumerate lattice points, or we only need to determine which horizontal rows are swept over by the features and to determine the range of x coordinates swept along each row.

6.4.1 Valid Orientations: $\Theta_{L/R}$

The term $\Theta_{L/R}$ refers to the set of possible object orientations capable of providing contact between the left-jaw specified edges and pegs on the left jaw, and the right-jaw specified edges and pegs on the right jaw. $\Theta_{L/R}$ expresses the constraint that there must be a point on a left-jaw specified edge which lies to the left of a point on a right-jaw specified edge.

It is important to note that, for generalized polygonal objects, it can be the case that one feature can intersect the convex hull of another; this case did not occur in the original algorithm which was restricted to linear features. In this case, the pair of features does not preclude any orientations with respect to left/right constraints.

6.4.2 Annulus Wedges

Given the set of possible orientation ranges $\Theta_{L/R}$, we use a conservative geometrical strategy to bound the set of possible peg positions. For each pair of features, the possible relative peg positions must correspond to vectors between points on the two features. This region can be conservatively characterized by wedges of an annulus, where the annulus wedges are characterized by a radius range and an orientation range.

Computing the annulus wedges for two linear features is simple because the extremal orientations and distances correspond to extremal points on the features. The minimum orientation, maximum orientation, and maximum distance all correspond to pairs of edge segment endpoints. The minimum distance might also correspond to the minimum distance between one endpoint and the other feature. For circular features, the computation of annulus wedges is slightly more involved because extremal orientations and distances may correspond to tangent points on the features.

6.4.3 Possible Second Peg Positions

The possible positions of the second peg $\{\mathcal{F}_j\}$ correspond to the lattice points within a section of an annulus. The orientation of the vector between $\mathcal{F}_i$ to $\mathcal{F}_j$ must lie within the Minkowski sum of $\Theta_{L/R}$ and the orientation ranges between points on $\mathcal{E}_i$ and $\mathcal{E}_j$ (Figure 6.8).

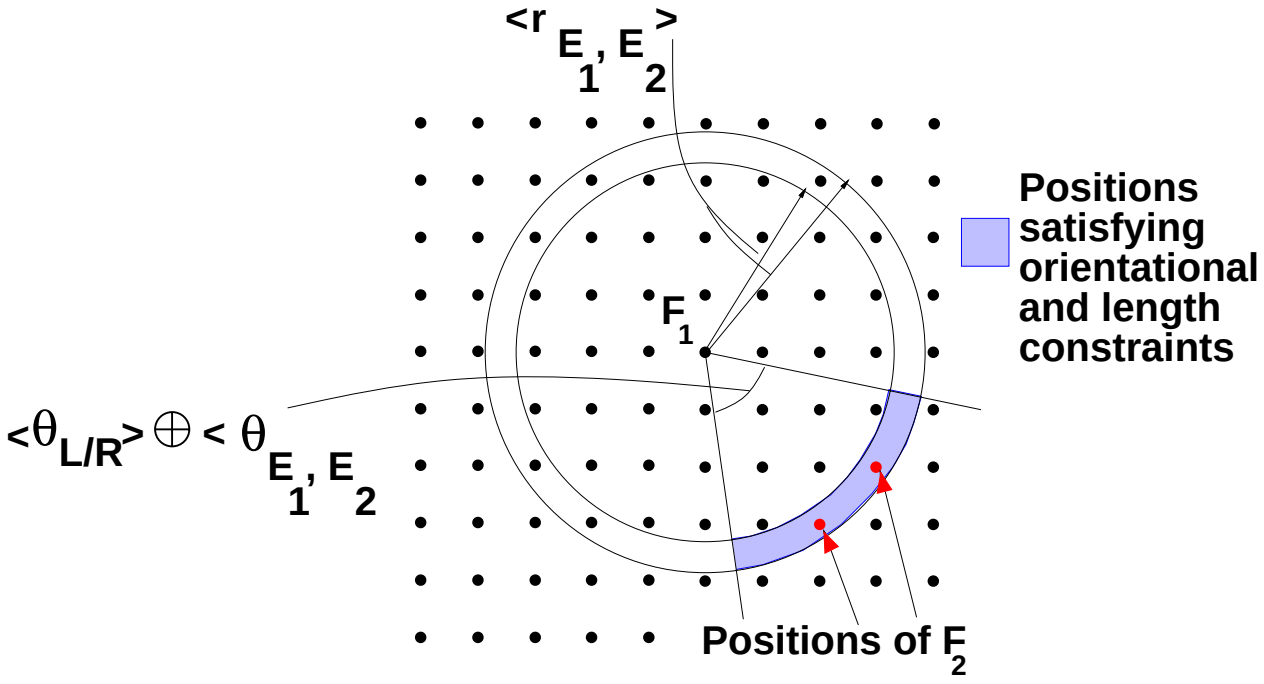


Figure 6.8: $\mathcal{F}_2$ is inside a wedge of the annulus defined by $(\langle r_{\mathcal{E}_1,\mathcal{E}_2}\rangle, \langle\theta_{L/R}\rangle \oplus \langle\theta_{\mathcal{E}_1,\mathcal{E}_2}\rangle)$.

6.4.4 Third Peg Positions for Type II Configurations

Enumerating the possible third peg positions for Type II configurations (where three pegs reside on one jaw, and one on the other) is relatively simple since the third peg lies on the same fixture plate as the first two, and is constrained to satisfy geometric constraints with respect to the first peg, and geometric constraints with respect to the second peg (Figure 6.10).

The third peg must lie within the intersection of the two respective wedges of annuli (Figure 6.9). We can compute the workpiece's pose from the first three contacts, and then enumerate the rows which contact the transformed fourth edge.

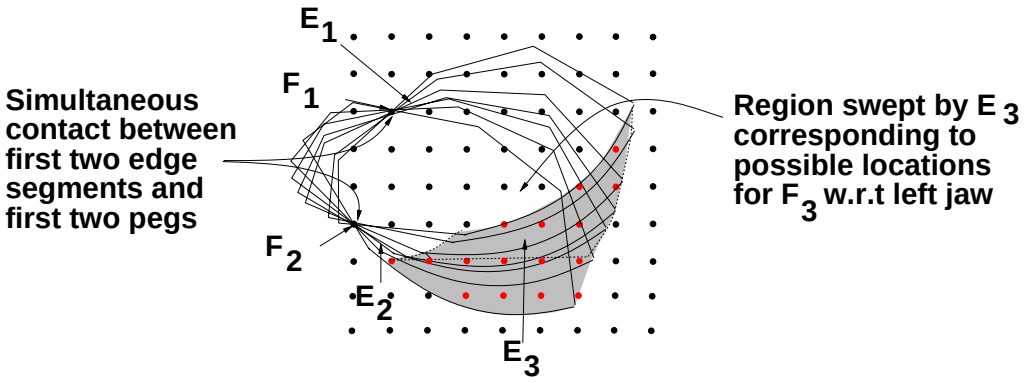


Figure 6.9: The third peg must lie in the region swept by $\mathcal{E}_3$ while maintaining contact between $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{F}_1$, $\mathcal{F}_2$; this region is defined with respect to the left jaw. Since for Type II configurations, the third peg lies on the same jaw as the first two pegs, the third peg must be a lattice point within the swept region.

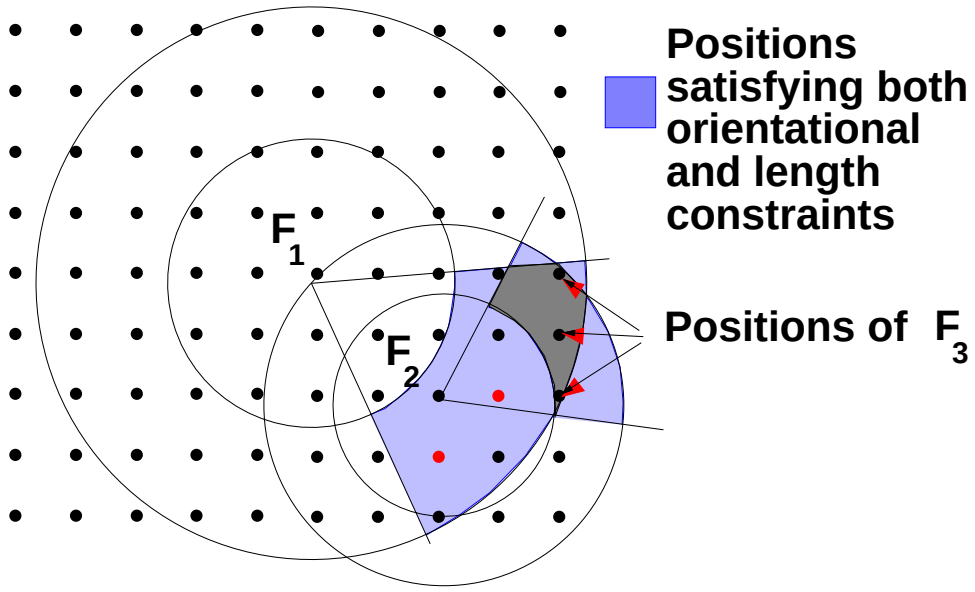


Figure 6.10: For Type II configurations, $\mathcal{F}_3$ is inside the intersections of two wedge annuli: $\langle r_{\mathcal{E}_1,\mathcal{E}_3} \rangle$ and $\langle \theta_{L/R} \rangle \oplus \langle \theta_{\mathcal{E}_1,\mathcal{E}_3} \rangle$, $\langle r_{\mathcal{E}_2,\mathcal{E}_3} \rangle$ and $\langle \theta_{L/R} \rangle \oplus \langle \theta_{\mathcal{E}_2,\mathcal{E}_3} \rangle$.

Third and Fourth Peg Positions for Type I Configurations

However, for Type I contacts, enumerating the positions of the third and fourth pegs is more difficult because they reside on the right vise jaw which may translate freely relative to the origin on the left vise jaw (Figure 6.11). The region swept by the third and fourth edges measured with respect to the origin on the left jaw only provides relative information about the possible x positions of the third and fourth fixture pegs. This is because all of the computations were performed with respect to the left jaw; although these regions cannot specify possible peg positions, the absolute y positions of the regions and the difference between the x positions of the regions does specify the possible peg positions. First we compute the regions (annulus wedges) for the third and fourth features with respect to the left jaw origin, and then the peg positions are bounded by the differentials between these regions.

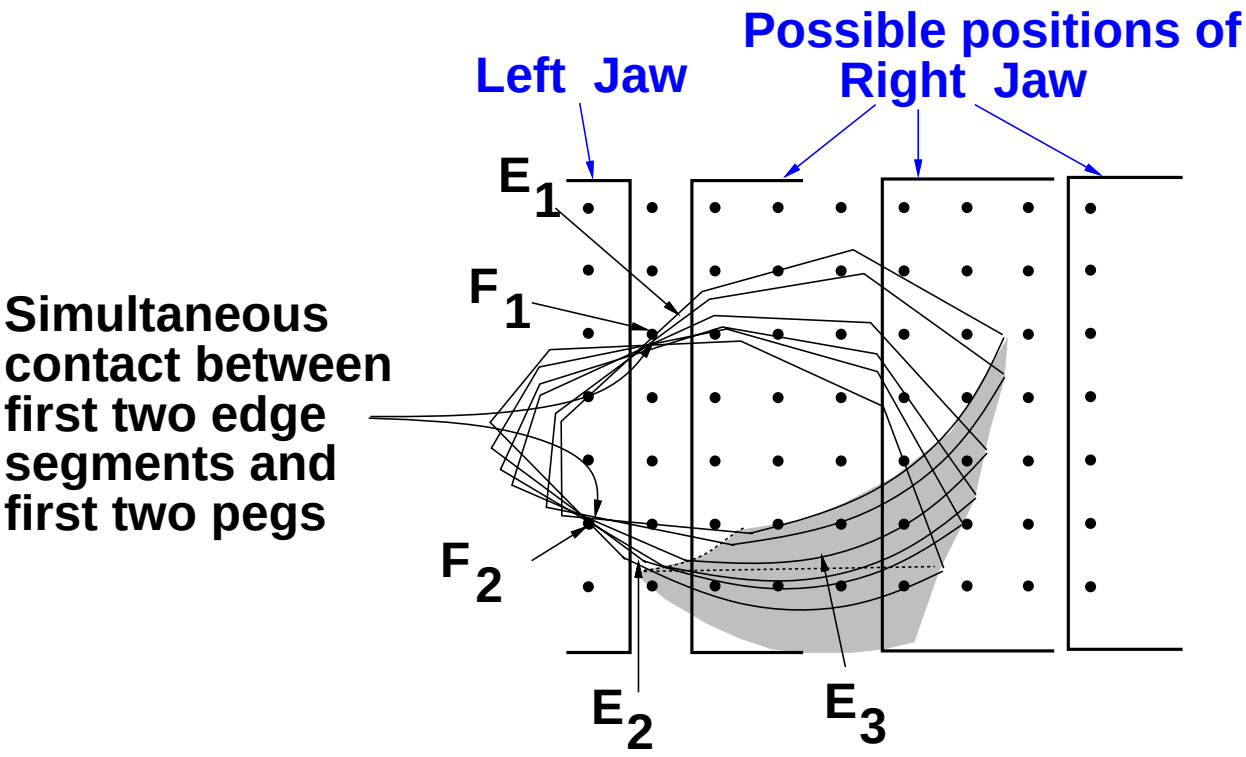


Figure 6.11: In Type I cases, the right jaw is free to translate, so the swept region indirectly specifies the set of possible positions for the third and fourth pegs.

The possible peg positions are enumerated from the regions as follows: Consider all

pairs of rows corresponding to the two edge segments, $(\mathcal{R}, \mathcal{R}^*)$, and the x regions included along those rows, $[X^{\mathcal{R}}_{\mathcal{E}_3}]$, $[X^{\mathcal{R}^*}_{\mathcal{E}_4}]$. Even though the x ranges $[X^{\mathcal{R}}_{\mathcal{E}_3}]$, $[X^{\mathcal{R}^*}_{\mathcal{E}_4}]$ were computed with respect to the left jaw, the differential between the x coordinate ranges $[X^{\mathcal{R}}_{\mathcal{E}_3}] \oplus (\Leftrightarrow [X^{\mathcal{R}^*}_{\mathcal{E}_4}])$ for $\mathcal{E}_3$ and $\mathcal{E}_4$ is invariant with respect to translation in x. We enumerate a pair of fixture peg positions $(\mathcal{F}_3, \mathcal{F}_4)$ for each discrete value $(k\lambda_x \in [X^{\mathcal{R}}_{\mathcal{E}_3}] \oplus (\Leftrightarrow [X^{\mathcal{R}^*}_{\mathcal{E}_4}])(k \in \mathbb{I}))$ in the range (Figure 6.12).

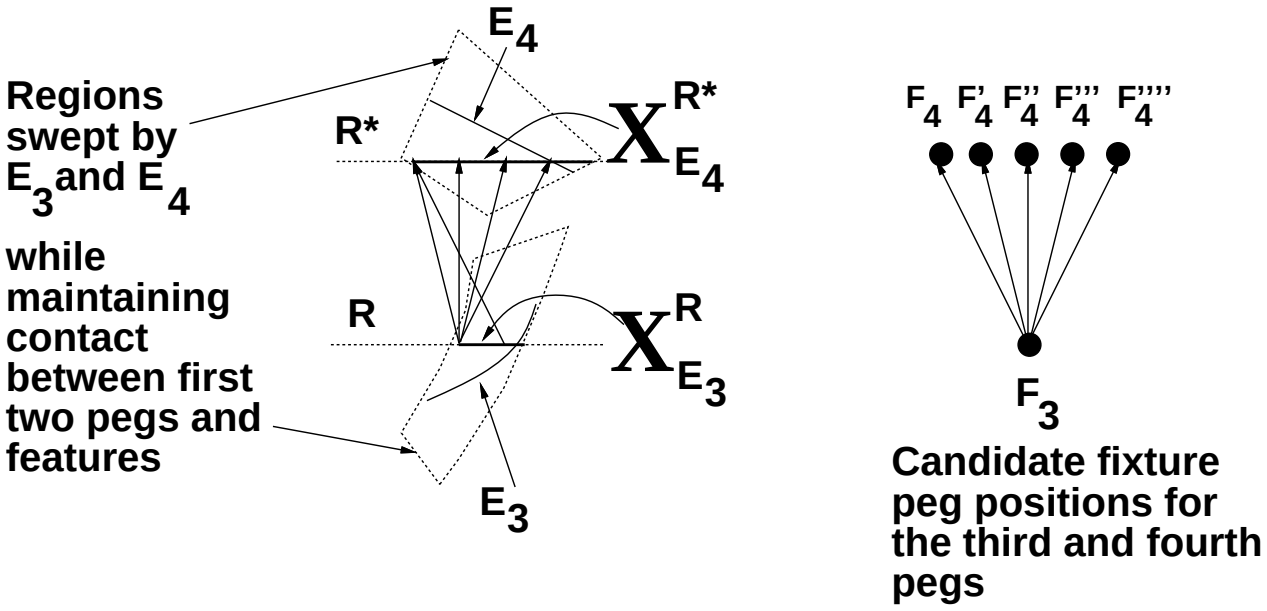


Figure 6.12: Possible fixture positions $\{(\mathcal{F}_3, \mathcal{F}_4), (\mathcal{F}'_3, \mathcal{F}'_4), \ldots\}$ are computed using the x coordinate ranges swept by the third and fourth edges $X^{\mathcal{R}}_{\mathcal{E}_3}, X^{\mathcal{R}}_{\mathcal{E}_4}$.

6.5 Computing Simultaneous Contact Poses

In this section, we describe a method for computing the object's poses which achieve simultaneous contact between the pegs and the model features when the jaws are free to translate in x. We solve for these poses using a technique similar to the original method described for polygonal models [WC94], where we computed simultaneous contact poses for four linear edges and four contact points using a combination of algebraic methods and numerical techniques [Man92; Dix08]. In this algorithm, as well as in the original algorithm, we formulated the contact constraints algebraically, yielding a nonlinear multivariate system (section 3.5.5). The approach we take is actually very similar to that taken in section 3.5.5:

for Type II cases, we use the same parameterization, and solve for the contact pose in the same manner (Dixon resultants). Notice that we are effectively solving the same problem in both situations; in Chapter 7, we will describe a duality between modular fixturing and scanning sensing.

The simultaneous contact pose corresponds to the pose which maps the model features onto the contact points. Since it is easier to characterize transformed points than transformed features, we will compute the transformation which maps the point set onto the model features, and then take the inverse.

One drawback of elimination approaches in particular, and algebraic approaches in general, is that the running time (and correspondingly, the imprecision) increases exponentially in the number of variables and polynomial degree. It is therefore desirable to characterize the system in as few variables as possible and with minimal algebraic complexity.

6.5.1 Parameterizations

In order to solve for the simultaneous contact poses, we use two related parameterizations which implicitly achieve contact between the characteristic point and the characteristic circular feature (any circular feature can be the characteristic feature, the characteristic point is simply the point associated with that feature). By implicitly satisfying one of the constraints, these parameterizations simplify the constraint formulation. In section 6.5.1, we discuss the pose parameterization used to solve for Type II cases (which was previously described in section 3.5.2), and in section 6.5.3, we discuss the pose parameterization used to solve for Type I cases.

(θ, ϕ) Parameterization: Type II

For Type II configurations, we use a parameterization (θ, ϕ) which corresponds to rotating the point set by θ and then translating the point set by $(R\cos(\phi), R\sin(\phi))$, where R refers to the radius of the characteristic circular feature. This parameterization implicitly achieves contact between the characteristic point and the characteristic circular feature assuming that both the characteristic point, and characteristic circle, are centered at the origin.

For Type II configurations, the simultaneous contact pose is found by determining

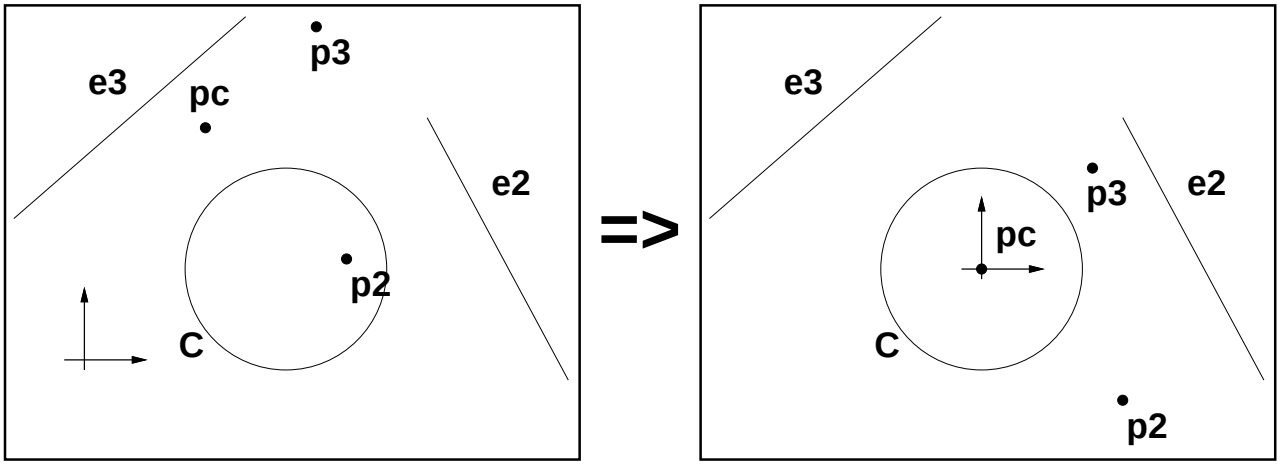


Figure 6.13: In order to ensure contact between the transformed characteristic point and the characteristic circular feature, both the Type I and Type II parameterizations require that the the center of the characteristic circle and the characteristic point both lie at the origin .

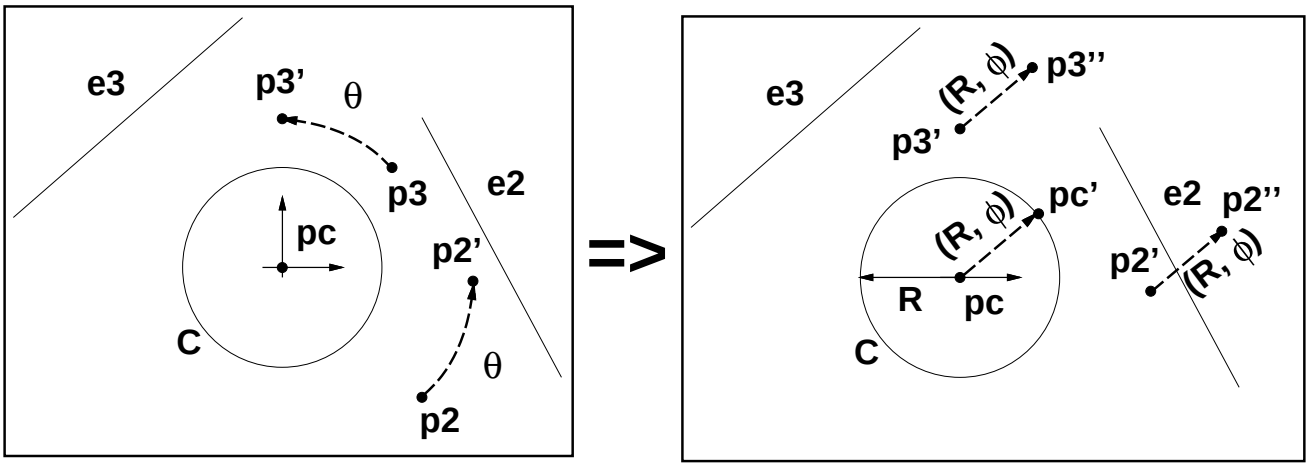


Figure 6.14: The Type II parameterization involves rotating the other 2 points around the origin by θ and then translating them by the vector expressed in polar coordinates as (R, ϕ).

the pose which achieves simultaneous satisfaction of three constraints corresponding to the three contact features.

Equation (6.1) presents the two-dimensional rigid transformation, in homogeneous coordinates, corresponding to (θ, ϕ). In order to arrive at algebraic expressions, we used the trigonometric substitutions $t = \tan(\frac{\theta}{2})$ and $u = \tan(\frac{\phi}{2})$.

$$\begin{bmatrix} \cos(\theta) & \Leftrightarrow\sin(\theta) & R\cos(\phi) \\ \sin(\theta) & \cos(\theta) & R\sin(\phi) \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} (1 \Leftrightarrow t^2)(1+u^2) & \Leftrightarrow(2t)(1+u^2) & R(1+t^2)(1 \Leftrightarrow u^2) \\ (2t)(1+u^2) & (1 \Leftrightarrow t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \quad (6.1)$$

6.5.2 Contact Constraints

The roots of $h(t,u)$ in equation (6.2) characterize the constraint that a point is mapped onto a line and the roots of $g(t,u)$ equation (6.3) characterize the constraint that a point is mapped onto a circle. In equation (6.2), the point is parameterized by (X,Y) and the line is parameterized by (a,b,c) corresponding to $a\mathtt{X} + b\mathtt{Y} = c$. In equation (6.3), the circle is parameterized by (xc, yc, r) corresponding to $(xc \Leftrightarrow X)^2 + (yc \Leftrightarrow Y)^2 = r^2$. Fortunately the resulting bi-quartic expression $g(t,u)$ is divisible by $(1+t^2)(1+u^2)$ yielding a biquadratic expression.

$$h(t,u) = \left(\begin{bmatrix} (1-t^2)(1+u^2) & -(2t)(1+u^2) & R(1+t^2)(1-u^2) \\ (2t)(1+u^2) & (1-t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \right)^T \cdot \begin{bmatrix} a \\ b \\ -c \end{bmatrix} \quad (6.2)$$

$$\begin{aligned} K(t,u) &= \left(\begin{bmatrix} (1-t^2)(1+u^2) & -(2t)(1+u^2) & R(1+t^2)(1-u^2) \\ (2t)(1+u^2) & (1-t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \right) \\ &\quad - \begin{bmatrix} xc \\ yc \\ (1+t^2)(1+u^2) \end{bmatrix} \\ g(t,u) &= K(t,u)^T \cdot K(t,u) - r^2((1+t^2)(1+u^2))^2 \end{aligned} \quad (6.3)$$

6.5.3 (θ, ϕ, σ) Parameterization: Type I

For Type I configurations, the simultaneous contact poses correspond to achieving simultaneous satisfaction of four contact constraints, where two of the points are free to translate in x. For this problem, we use a similar parameterization which has an additional degree of freedom, the jaw separation σ (Figure 6.15). The contact constraint for the second feature (which resides on the left vise jaw) is the same as the one given for Type II configurations. The contact constraints for the third and fourth features (which both reside on the right vise jaw) involve σ. Since all four contact constraints are necessary to constrain the object's pose, computing the simultaneous contact pose involves intersecting three constraints.

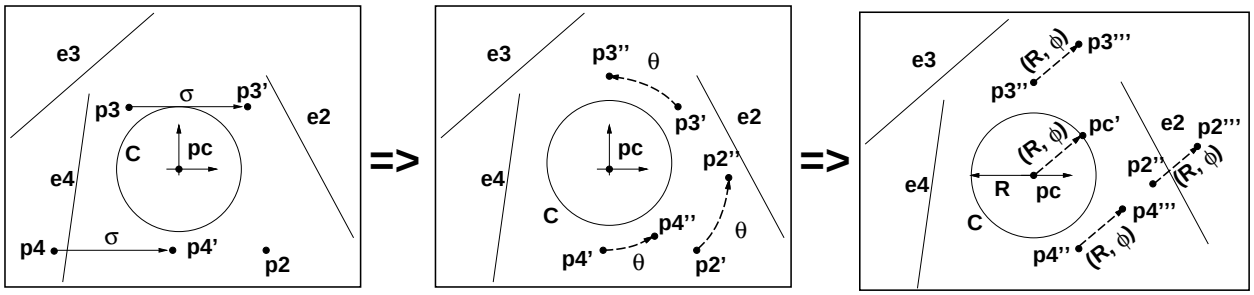


Figure 6.15: The Type I parameterization involves translating two of the points in x by σ and then rotating the 3 points around the origin by θ and then translating them by the vector expressed in polar coordinates as (R, ϕ).

The roots of $h_\sigma(t, u)$ in equation (6.4) express the constraint that a point (X, Y) is transformed onto a line parameterized by (a, b, c) corresponding to $a\mathtt{X} + b\mathtt{Y} = c$ where the point (X, Y) is first translated in x by σ.

$$h_\sigma(t,u) = \left(\begin{bmatrix} (1-t^2)(1+u^2) & -(2t)(1+u^2) & R(1+t^2)(1-u^2) \\ (2t)(1+u^2) & (1-t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X+\sigma \\ Y \\ 1 \end{bmatrix} \right)^T \cdot \begin{bmatrix} a \\ b \\ -c \end{bmatrix} \quad (6.4)$$

The roots of $g_\sigma(t, u)$ in equation (6.5) express the constraint that a point (X, Y) is transformed onto a circle parameterized by (xc, yc, r) corresponding to $(xc \Leftrightarrow X)^2 + (yc \Leftrightarrow Y)^2 = r^2$ where the point (X, Y) is first translated in x by σ. Fortunately the resulting bi-

quartic expression $g_\sigma(t,u)$ is divisible by $(1+t^2)(1+u^2)$ producing a biquadratic expression.

$$
\begin{aligned}
K_\sigma &= \left(\begin{bmatrix} (1-t^2)(1+u^2) & -(2t)(1+u^2) & R(1+t^2)(1-u^2) \\ (2t)(1+u^2) & (1-t^2)(1+u^2) & R(1+t^2)(2u) \\ 0 & 0 & (1+t^2)(1+u^2) \end{bmatrix} \begin{bmatrix} X+\sigma \\ Y \\ 1 \end{bmatrix} \right) \\
&\quad - \begin{bmatrix} xc \\ yc \\ (1+t^2)(1+u^2) \end{bmatrix} \\
g_\sigma(t,u) &= K_\sigma(t,u)^T \cdot K_\sigma(t,u) - r^2((1+t^2)(1+u^2))^2 \qquad (6.5)
\end{aligned}
$$

6.5.4 Example

In this section, we step through an example to clearly explain the the simultaneous contact pose computation. In this example, we compute the simultaneous contact pose for four points corresponding to two circular features and two linear features.

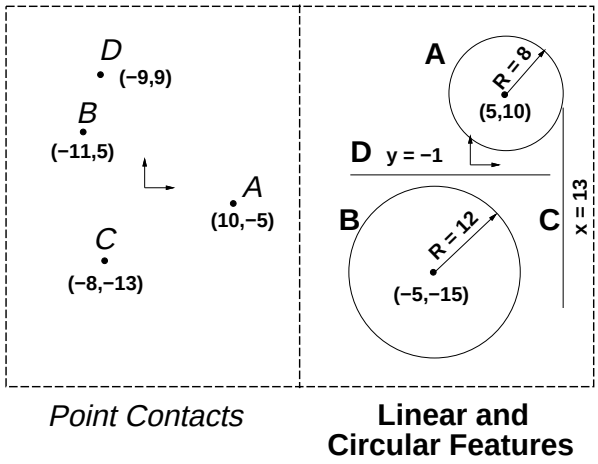


Figure 6.16: Four points and features for which we will compute the pose achieving simultaneous contact, where two of the four points are free to translate in x.

In this example, point $A = (10, \Leftrightarrow 5)$ should contact circle **A** centered at $(5, 10)$ with radius 8. Point $B = (\Leftrightarrow 11, 5)$ should contact circle **B** centered at $(\Leftrightarrow 5, \Leftrightarrow 15)$ with radius 12. Point $C = (\Leftrightarrow 8, \Leftrightarrow 13)$ should contact the vertical line **C**, defined by $x = 13 (a = 1, b = 0, c = 13)$. Point $D = (\Leftrightarrow 9, 9)$ should contact the horizontal line **D** defined by $y = \Leftrightarrow 1 (a = 0, b = 1, c = \Leftrightarrow 1)$. One transformation which achieves simultaneous contact

between the points and features maps A to $(5, 18)$, B to $(\Leftrightarrow 5, \Leftrightarrow 3)$, C to $(13, 0)$, and D to $(\Leftrightarrow 9, \Leftrightarrow 1)$.

First we translate the point and features sets so that the first point and the center of the first circle both lie at the origin (as shown in Figure 6.17). Point A is translated to $(0, 0)$, point B is translated to $(\Leftrightarrow 21, 10)$, point C is translated to $(\Leftrightarrow 18, \Leftrightarrow 8)$, and point D is translated to $(\Leftrightarrow 19, 14)$. Circle **A** is translated to $(0, 0)$, circle **B** is translated to $(\Leftrightarrow 10, \Leftrightarrow 25)$, line **C** is translated to $x = 8$, and line **D** is translated to $y = \Leftrightarrow 11$.

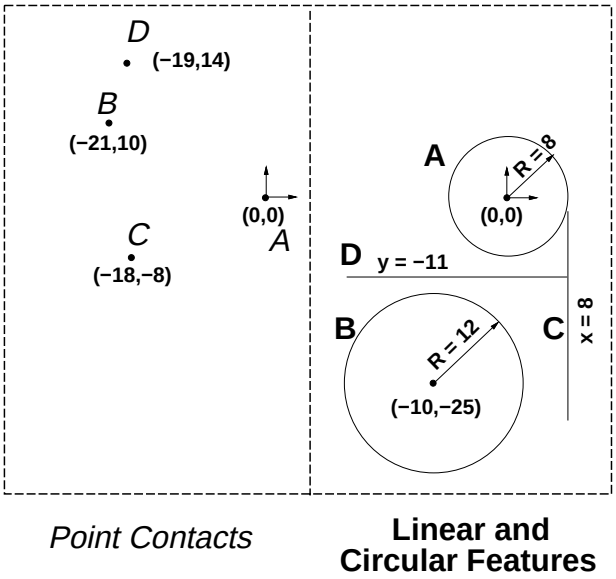


Figure 6.17: Transform the point sets and feature sets so that the first point and the center of the first circle both lie at the origin.

Equation (6.6) corresponds to the constraint that point B is mapped onto circular feature B. Equation (6.7) corresponds to the constraint that point C is mapped onto linear feature C. Equation (6.8) corresponds to the constraint that point D is mapped onto linear feature D. Next, we separate $f_1(t, u)$, $f_2(t, u)$, and $f_3(t, u)$ into their u coefficients (equations (6.9)-(6.23)).

$$f_1(t, u | R = 8, X = \Leftrightarrow 21, Y = 10, xc = \Leftrightarrow 10, yc = \Leftrightarrow 25, rc = 12^2) = \tag{6.6}$$
$$1090 \Leftrightarrow 2820\, t + 1602\, t^2 + 1120\, u \Leftrightarrow 1344\, t\, u + 480\, t^2\, u + 1442\, u^2 \Leftrightarrow 2180\, t\, u^2 + 610\, t^2\, u^2$$
$$f_2(t, u | R = 8, X = \Leftrightarrow 18, Y = \Leftrightarrow 8, a = 1, b = 0, c = 8) = \tag{6.7}$$
$$\Leftrightarrow 18 + 16\, t + 18\, t^2 \Leftrightarrow 34\, u^2 + 16\, t\, u^2 + 2\, t^2\, u^2 + X \Leftrightarrow t^2 X + u^2 X \Leftrightarrow t^2 u^2 X$$

$$f_3(t,u|R=8,X=\Leftrightarrow 19,Y=14,a=0,b=1,c=\Leftrightarrow 11) = \tag{6.8}$$
$$\Leftrightarrow 3 \Leftrightarrow 38\,t + 25\,t^2 + 16u + 16t^2u \Leftrightarrow 3\,u^2 \Leftrightarrow 38\,t\,u^2 + 25\,t^2\,u^2 + 2tX + 2tu^2X$$

$$f_{1,0,0}(t) = 1602t^2 \Leftrightarrow 2820t + 1090 \tag{6.9}$$
$$f_{1,1,0}(t) = 480t^2 \Leftrightarrow 1344t + 1120 \tag{6.10}$$
$$f_{1,2,0}(t) = 610t^2 \Leftrightarrow 2180t + 1442 \tag{6.11}$$
$$f_{2,0,0}(t) = 18t^2 + 16t \Leftrightarrow 18 \tag{6.12}$$
$$f_{2,1,0}(t) = 0 \tag{6.13}$$
$$f_{2,2,0}(t) = 2t^2 + 16t \Leftrightarrow 34 \tag{6.14}$$
$$f_{2,0,1}(t) = 1 \Leftrightarrow t^2 \tag{6.15}$$
$$f_{2,1,1}(t) = 0 \tag{6.16}$$
$$f_{2,2,1}(t) = 1 \Leftrightarrow t^2 \tag{6.17}$$
$$f_{3,0,0}(t) = \Leftrightarrow 3 \Leftrightarrow 38t + 25t^2 \tag{6.18}$$
$$f_{3,1,0}(t) = 16 + 16t^2 \tag{6.19}$$
$$f_{3,2,0}(t) = \Leftrightarrow 3 \Leftrightarrow 38t + 25t^2 \tag{6.20}$$
$$f_{3,0,1}(t) = 2t \tag{6.21}$$
$$f_{3,1,1}(t) = 0 \tag{6.22}$$
$$f_{3,2,1}(t) = 2t \tag{6.23}$$

Next, we compute the resultant matrix $M(t)$ using the Dixon formulation (equation (6.24)). We use a companion matrix formulation to solve for the roots of t such that $|M(t)| = 0$.

$$M(t) = \begin{bmatrix} M_{00}(t) & M_{01}(t) \\ M_{10}(t) & M_{11}(t) \end{bmatrix} \tag{6.24}$$

$$M_{00}(t) = f_{1,1,0}f_{2,0,1}f_{3,0,0} - f_{1,1,0}f_{2,0,0}f_{3,0,1} + f_{1,0,0}f_{2,1,0}f_{3,0,1} - f_{1,0,0}f_{2,0,1}f_{3,1,0}$$

$$M_{01}(t) = f_{1,2,0}f_{2,0,1}f_{3,0,0} - f_{1,2,0}f_{2,0,0}f_{3,0,1} + f_{1,0,0}f_{2,2,0}f_{3,0,1} - f_{1,0,0}f_{2,0,1}f_{3,2,0}$$

$$M_{10}(t) = f_{1,2,0}f_{2,0,1}f_{3,0,0} - f_{1,2,0}f_{2,0,1}f_{3,0,0} + f_{1,1,0}f_{2,1,1}f_{3,0,0} - f_{1,1,0}f_{2,0,0}f_{3,1,1} +$$
$$f_{1,0,0}f_{2,2,0}f_{3,0,1} - f_{1,0,0}f_{2,0,1}f_{3,2,0} + f_{1,0,0}f_{2,1,1}f_{3,1,0} - f_{1,0,0}f_{2,1,0}f_{3,1,1}$$

$$M_{11}(t) = f_{1,2,0}f_{2,1,1}f_{3,0,0} - f_{1,2,0}f_{2,0,0}f_{3,1,1} + f_{1,2,0}f_{2,0,1}f_{3,1,0} - f_{1,2,0}f_{2,1,0}f_{3,0,1} +$$
$$f_{1,1,0}f_{2,2,0}f_{3,0,1} - f_{1,1,0}f_{2,0,1}f_{3,2,0} + f_{1,0,0}f_{2,2,0}f_{3,1,1} - f_{1,0,0}f_{2,1,1}f_{3,2,0}$$

$$
\begin{aligned}
M_{00}(t) &= -27072t^6 + 48192t^5 + 10688t^4 - 81920t^3 + 78144t^2 - 9280t - 10560 \\
M_{01}(t) &= 2976t^6 + 51328t^5 - 152096t^4 + 121856t^3 - 43040t^2 + 19584t - 8800 \\
M_{10}(t) &= 2976t^6 + 51328t^5 - 152096t^4 + 121856t^3 - 43040t^2 + 19584t - 8800 \\
M_{11}(t) &= 11200t^6 - 22592t^5 - 48064t^4 + 133120t^3 - 105280t^2 + 34880t + 4928
\end{aligned}
$$

There are four roots of $t = \{0, 1.845457, 1.482830, 0.690271, 1.0\}$ which satisfy the condition $|M(t)| = 0$. These roots correspond to the following transformations: $(X = 0, Y = 0, \theta = \frac{-\pi}{2}, \sigma = 0)$, $(X = \Leftrightarrow 6.617315, Y = 1.944347, \theta = \Leftrightarrow 1.208333, \sigma = 1.362430)$, $(X = 10.501454, Y = 2.115092, \theta = \Leftrightarrow 1.954937, \sigma = \Leftrightarrow 21.205877)$, $(X = \Leftrightarrow 3.642182, Y = 4.361851, \theta = \Leftrightarrow 2.148431, \sigma = \Leftrightarrow 7.187267)$. All of these planar transformations achieve simultaneous contact between the two circular features and two linear features and the four points, where two of the points are initially free to translate in x. Of course, valid transformations may not correspond to fixture configurations because they do not exhibit force closure, or because they rely involve unrealizable jaw separations σ ($\sigma < 0$).

6.6 Results

In this section, we present examples of fixtures designed by the algorithm, and quantitative results characterizing the number of fixtures found for various workpieces. Table 6.1 lists the number of fixture configurations found for various objects, for various λ_y, λ_x spacings, various lattice sizes (characterized by the number of columns), and the time spent computing the configurations. We focus on single column fixtures because they can be easily manufactured, and can provide a universal gripper by equipping a parallel jaw gripper with pneumatically actuated pegs. The fixture counts already discount object and lattice symmetries by only counting one fixture for each family of translated or symmetric fixtures.

6.6.1 Example Fixtures

Figures 6.18-6.24 show fixtures generated by our algorithm.

Object	λ_y, λ_x	# Columns	# Type I	# Type II	Total	Time (seconds)
thin4Handle	10,10	1	208	4	212	792
8Handle	10,10	1	36	0	36	356
small4Handle	10,10	1	14	0	14	395
small4Handle	10,10	10	88	123	211	9195
large4Handle	10,10	1	40	5	45	949
large4Handle	10,10	10	492	572	1064	38968
large4Handle	8,8	10	1394	1654	3048	
oblongPart	10,10	1	26	0	26	112
oblongPart	10,10	10	280	198	478	4217
oblongPart	8,8	10	801	278	1079	13896
widget	10,10	1	305	62	367	8644
widget	8,8	1	636	124	760	8896
flange	10,10	1	60	1	61	1742

Table 6.1: Total Numbers of Suitable Fixture Configurations

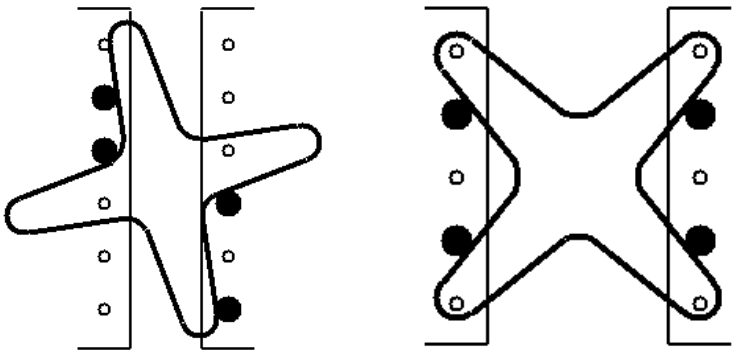
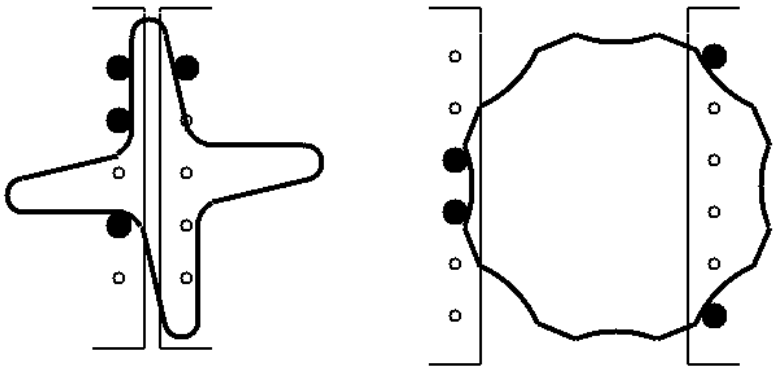
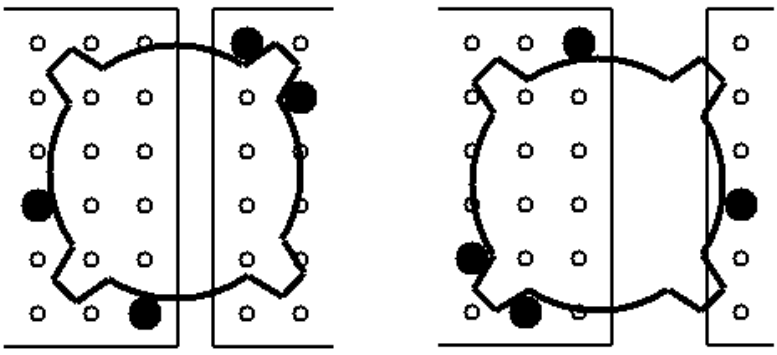


Figure 6.18: Two Type I fixtures for thin4Handle, a thin four sided handle



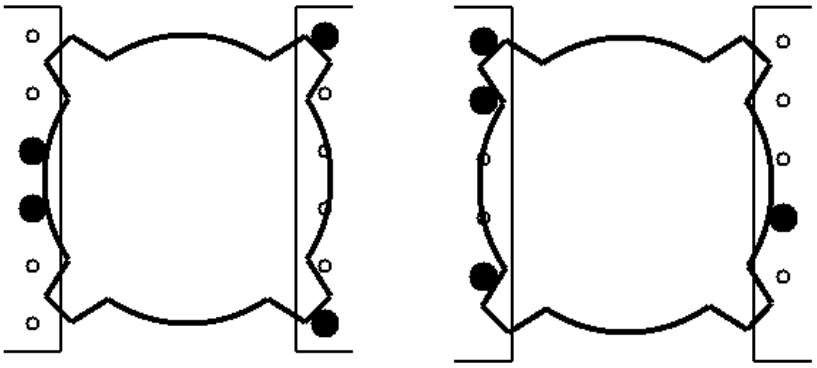
(a) (b)

Figure 6.19: (a) a Type II fixture for thin4Handle (b) a Type I fixture for 8Handle, an eight sided handle



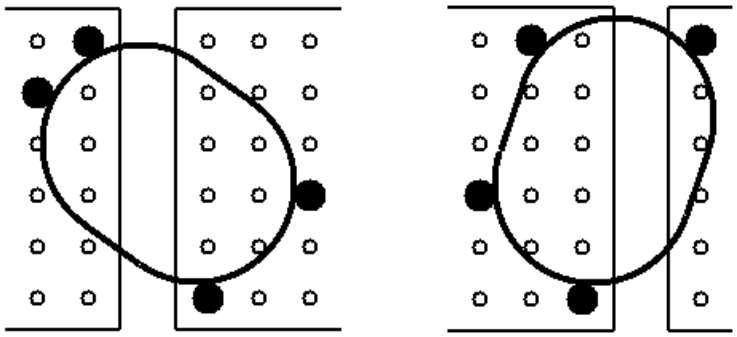
(a) (b)

Figure 6.20: (a) a Type I fixture for small4Handle, a small four sided handle (b) a Type II fixture for small4Handle



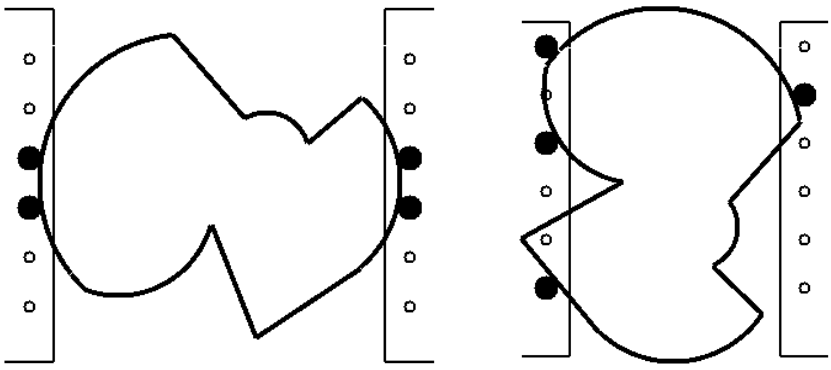
(a) (b)

Figure 6.21: (a) a Type I fixture for large4Handle, a large four sided handle (b) a Type II fixture for large4Handle



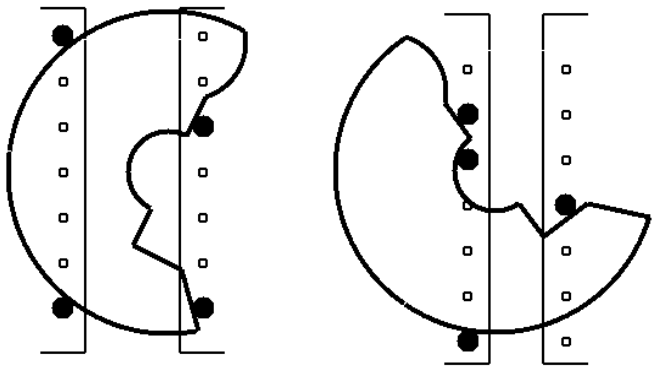
(a) (b)

Figure 6.22: (a) a Type I fixture for oblongPart, an oblong cap (b) a Type II fixture for oblongPart



(a) (b)

Figure 6.23: (a) a Type I fixture for widget (b) a Type II fixture for widget



(a) (b)

Figure 6.24: (a) a Type I fixture for flange (b) a Type II fixture for flange

6.6.2 Degrees of Resultant Polynomials

Since we use numerical techniques to compute the simultaneous contact poses, we are very interested in the degrees of the corresponding univariate polynomials because the running time and imprecision of the solver routines increase with the polynomial degree. Table 6.2 presents the polynomial degrees corresponding to solving for θ for various scenarios; after solving for θ, the translation parameters (x, y) can be solved for in a straightforward manner, since the resulting problem is linear.

6.7 Conclusion

In this chapter, we described a complete, efficient algorithm for designing fixture vise configurations for generalized polyhedral prismatic workpieces. This fixture vise has the advantageous property that there are only a finite number of ways to fixture an arbitrary object, therefore, fixtures can be enumerated in a generate and test manner. In this chapter, we extended the algorithm for designing fixtures for arbitrary prismatic polyhedral workpieces, and described the extensions involved in handling generalized polyhedral prismatic workpieces. Extending the algorithm mainly involves two key steps: enumerating all peg configurations which simultaneously contact four edge segments, and for each such configuration, computing the contact points and pose of the workpiece. We compute the simultaneous contact poses algebraically, and to reduce the complexity of the constraints, we utilize a custom parameterization which implicitly satisfies one of the contact constraints.

Type	# Left Linear Features	# Left Circular Features	# Right Linear Features	# Right Circular Features	Max. Polynomial Degree
I	2	0	2	0	4
I	1	1	2	0	12
I	0	2	2	0	12
I	1	1	1	1	24
I	1	1	0	2	24
I	0	2	0	2	24
II	3	0	1	0	2
II	3	0	0	1	2
II	2	1	1	0	4
II	2	1	0	1	4
II	1	2	1	0	6
II	1	2	0	1	6
II	0	3	1	0	8
II	0	3	0	1	8

Table 6.2: The maximum degree of the resultant expressions utilized for solving for each case

Chapter 7

Duality Between Modular Fixturing and Scanning Sensing

Abstract

Fixturing, the act of immobilizing a workpiece for manufacturing or assembly operations, is a fundamental task in manufacturing. Fixtures can be fabricated from scratch or assembled from a toolkit of modular components; the latter approach is termed modular fixturing. Recently developed algorithms automatically design fixtures for arbitrary objects. In this chapter, we generalize these results into a generic strategy for enumerating modular fixture configurations for arbitrary workpieces and arbitrary fixture toolkits, such that the fixture toolkit consists of fixture elements which are inserted into a lattice of holes, and where the total number of contacts is equal to the total number of degrees of freedom. For these types of toolkits, only a finite number of configurations achieve simultaneous contact between the workpiece and the fixture elements, and, therefore, a generate and test approach suffices. Our generic fixture design algorithm follows this approach and generates configurations which achieve simultaneous contact between the object and the fixture, and then tests for force closure. This generic algorithm is based upon a duality between modular fixturing and scanning sensing; it turns out that every modular fixture configuration is analogous to a scanning sensing situation. Our generic fixture design algorithm corresponds to a generic approach for identifying objects from scanning sensors.

7.1 Introduction

Fixturing encompasses the design and assembly of fixtures to locate and hold a workpiece during a manufacturing operation such as machining or assembly. Fixtures can be fabricated from scratch, or assembled from a toolkit of precisely manufactured, mass produced fixture elements; the latter approach is termed *modular fixturing*. In the past, fixture design was more of a craft than a science, and most automated fixture design systems used expert systems to emulate a skilled machinist; unfortunately, without geometric analysis, expert systems are only capable of describing "types" of fixturing components, rather than computing fixture designs.

Many systems have been developed to automatically design fixtures, and some of these systems involve reconfigurable fixtures in which modular fixture elements are inserted into a lattice of holes. Recently, Wallack and Canny [WC94] (Chapters 6,7, Figure 7.1) and Brost and Goldberg [BG94] developed fixture design algorithms for related modular fixture toolkits. Both of these systems enumerated fixture designs for arbitrary workpieces for toolkits in which fixture elements were inserted into a lattice of holes, and where the total number of contacts was equal to the total number of degrees of freedom. In this chapter, we generalize these results and present a generic algorithm to automatically design fixtures for arbitrary workpieces and these types of modular fixture toolkits.

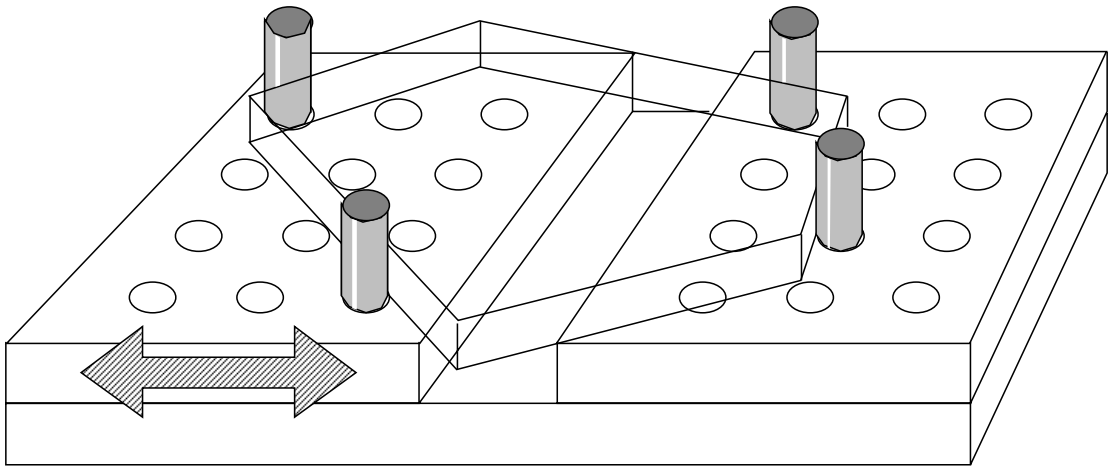


Figure 7.1: Wallack and Canny's fixture vise.

We use the term *minimal fixture toolkit* to refer to a fixture toolkit which relies on a single degree of freedom. At least one degree of freedom is necessary to accommodate

workpiece variations. In a frictionless point contact model, each contact provides a positive wrench in wrench space, and $k+1$ contacts (wrenches) are necessary to positively span a k dimensional space, *i.e.* achieve force closure. Therefore, any fixture toolkit must contain at least one degree of freedom in order to generically satisfy $k+1$ constraints. When the fixture only contains a single degree of freedom, the number of degrees of freedom is equal to the number of constraints, and there are only a finite number of simultaneous contact configurations (*i.e.,* fixtures).

In this chapter, we reexamine both Wallack and Canny's and Brost and Golberg's fixturing systems, emphasizing that both systems attained success because they focused on fixture toolkits for which there were always a finite number of solutions, so that simple generate and test strategies sufficed. We also observed a duality between modular fixture design for minimal toolkits and scanning sensing. This duality not only yields a generic fixture design algorithm for arbitrary workpieces and arbitrary minimal modular fixturing toolkits, but it enables knowledge transfer between domains.

Our generic fixture algorithm uses a generate and test approach since minimal fixture toolkits only admit a finite number of simultaneous contact configurations. First, we generate all fixture configurations which achieve simultaneous contact between the object and the fixture elements, and then we test these simultaneous contact configurations for force closure. We reduce the task of enumerating simultaneous contacts to the problem of intersecting curves in configuration space; the curves can be efficiently intersected using a sweep algorithm. Since the performance of this strategy depends upon the dimension of the configuration space, we also discuss methods for realizing a dense parameterization; *i.e.,* a parameterization in the fewest variables and with minimal complexity. Additionally, since in many applications, only a single fixture configuration is necessary, we also present heuristics for efficaciously enumerating fixture configurations.

7.1.1 Previous Work

Due to the duality between modular fixturing and scanning sensing, this research is also related to the problem of identifying objects from scanning sensor data via indexing; indexing is a technique for recognizing objects by distilling coordinates from sensed feature groups, and then quantizing these coordinates to produce integral indices to index a table entry containing the consistent interpretations. For correctness, indexing tables need to

contain entries for every possible combination of discretized coordinates and interpretations. In this chapter, we show that every modular fixture configuration corresponds to a separate indexing table entry, and therefore, a complete indexing table contains all of the valid fixture configurations. In this thesis, we have described approaches for constructing complete indexing tables [WC95a; Wal95b] (Chapters 2,3,8) which involved enumerating indexing table entries by enumerating cells in an arrangement in configuration space.

7.1.2 Outline

In section two, we describe a complete, efficient, algorithm for enumerating fixture configurations by enumerating simultaneous contact configurations and testing for force closure; in section three, we describe the duality between modular fixturing and scanning sensing; in section four, we present three new minimal modular fixturing toolkit hardware designs; in section five, we present some heuristics for ordering the enumeration of fixture designs to efficaciously enumerate fixture designs; we conclude by summarizing the results and advantages of these techniques.

7.2 Generic Design Algorithm for Minimal Fixture Toolkits

In this chapter, we describe a generic algorithm for designing fixtures for arbitrary workpieces and arbitrary minimal toolkits. Since minimal toolkits only provide a finite number of valid fixtures, the fixture configurations can be enumerated via a generate and test strategy. Basically, our generic algorithm generates all simultaneous contact poses, and then tests each simultaneous contact poses for force closure. The generate step involves enumerating all simultaneous contact configurations, *i.e.*, all positions of the fixture elements, fixture articulation, and object poses which achieve l simultaneous contacts, where l is the number of peg contacts.

First, we enumerate configurations of possible fixture elements. Without further analysis, we can always constrain the positions of the fixture elements because two fixture elements can never be separated by more than the maximum distance between two points on the object.

We can characterize each feature/fixture contact by a hypersurface in configuration space. If each model feature is defined as a surface patch, then the contact constraint

corresponds to a region of a hypersurface, where the hypersurface has codimension one. The region of the hypersurface is bounded is bounded by hypersurfaces of codimension two which characterize the constraint that the contact location lies within the surface patch associated with the feature.

Simultaneous contact configurations are produced by enumerating all intersections of l constraint hypersurfaces. We can ignore intersections which correspond to a higher dimensional subspaces than a point set, because they do not provide repeatable positioning. Thereby, we only need to consider a finite number of discrete configurations.

The caveat of this approach is that we need to carefully define equivalence classes for fixture configurations. If two *different* fixture configurations are analogous to each other, we only want to enumerate a single canonical description. For the fixture vise, equivalent configurations can be produced by translating the fixture elements and object up or down by one row, or left or right by one column.

We can highlight equivalence classes of fixtures by identifying redundant or irrelevant degrees of freedom, *i.e.,* uncovering a lower dimensional subspace of configuration space which contains all of the interesting characteristics of the larger configuration space. For example, for planar fixtures, a prismatic (swept-volume) polyhedron is characterizable by its the two-dimensional projection. For Brost and Goldberg's translating clamp toolkit, since the configuration is computable after choosing the first three peg contacts, they used a three dimensional (x, y, θ) configuration space, and ignored the translating clamp until later.

7.3 Duality Between Minimal Fixture Design and Complete Indexing Table Construction

In this section, we describe a duality between designing modular fixtures and recognizing objects using scanning sensors. We describe the reduction from modular fixture configurations to scanning sensing configurations. As a consequence of this duality, the task of enumerating modular fixture designs for minimal fixture toolkits is reducible to the task of constructing complete indexing tables for scanning sensing. Consequently, both problems can be solved by enumerating cells in an arrangement in configuration space.

This duality allows us to transfer knowledge from one domain to another, and

provides a different perspective which can highlight equivalence classes of fixtures. In other words, if the dual system can be parameterized by fewer degrees of freedom, then it must be the case that the fixture parameterization incorporates additional degrees of freedom which characterize equivalent fixtures. Thereby, comparing the parameterizations of the two dual systems highlights equivalence classes.

7.3.1 Scanning Sensing

Scanning sensor data can be produced by moving an object relative to a scanning sensor where the sensor corresponds to an array of beam sensors (Chapter 3). The positions where the sensor's output change are termed scanline endpoints (Figure 7.2). Given the scanline endpoint data and a set of modeled objects, scanning sensing involves identifying the scanned object and estimating its pose (refer Figure 7.3).

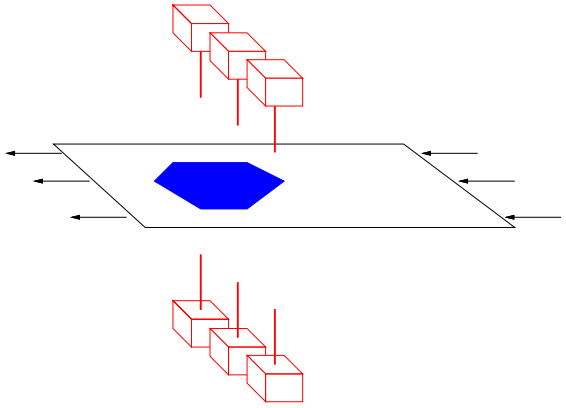


Figure 7.2: Scanning procedure

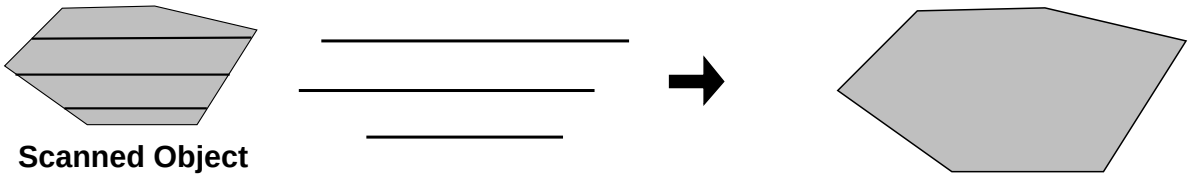


Figure 7.3: Given the scanline endpoints produced by scanning an arbitrary object, scanning sensing involves determining the object's identity and pose

7.3.2 Duality Between Fixture Vise and Horizontal Scanning

Figure 7.4 depicts the relationship between scanning sensing and modular fixturing. The dual of the fixture vise system is a scanning system which uses parallel scanning lines. Essentially, when certain criteria are satisfied, every modular fixture design corresponds to some scanning sensing configuration. For the fixture vise configuration in Figure 7.4, the four contact points are separated in y by integral multiples of λ_y $(k_1\lambda_y | k_1 \in \mathbb{I})$, and pairs of contact points are separated by integral multiples of λ_x $(k_2\lambda_x, k_3\lambda_x | k_2, k_3 \in \mathbb{I})$. These constraints can also be used to characterize a scanning configuration, where the scanlines are placed at integral multiples of λ_y $(k_1\lambda_y | k_1 \in \mathbb{I})$, and where two pairs of scanline endpoints are separated by integral multiples of λ_x $(k_2\lambda_x, k_3\lambda_x | k_2, k_3 \in \mathbb{I})$.

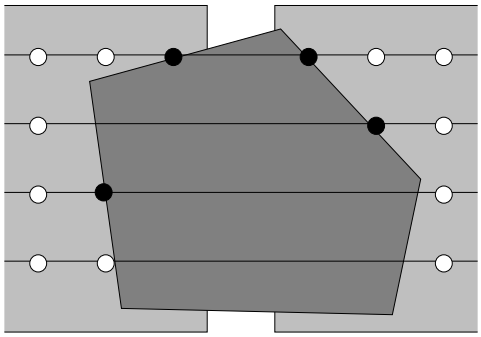
The reason behind this duality is that minimal fixtures use regular lattices and a single continuous degree of freedom. Although there are many methods to enumerate points in lattices, the continuous degree of freedom poses a more difficult problem. The scanning paradigm handles the lattices by considering a finite number of lines covering the lattices, and it also handles degree of freedom via scanning paradigm by considering the differences between pairs of intersections. By taking the differences between scan intersections, we have effectively normalized away the degree of freedom, leaving us with the problem of identifying pairs of points which differ by integral multiples of λ_x.

The general case uses non-parallel scanlines corresponding to lattice rows, and considers pairwise differences between scan intersections. In Figure 7.5, the eight scan intersection are defined in terms of the distance parallel to the scanline from the intersection to the object's reference point. In Figure 7.5, four our of the eight scan intersection distances are characterized by variables X_1, X_2, X_3, X_4; since there is an analogous modular fixture, then it must be the case that $X_1 \Leftrightarrow X_3 = k_1\lambda_x$ and $X_2 \Leftrightarrow X_4 = k_1\lambda_y$.

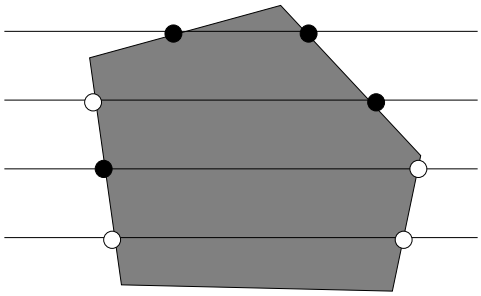
Since each fixture configuration corresponds to some scanning situation, then since there is a generic method for enumerating equivalence classes of scanning configurations, then we can use this method to enumerate generic modular fixture designs.

7.3.3 Indexing

Indexing is a constant time technique for determining which model features correspond to a set of sensed features, termed the correspondence problem (Chapters 2,3,8,9). Indexing techniques involve extracting indexing coordinates from tuples of sensed features,



2D View of Modular Fixture Configuration Achieving Simultaneous Contact



Scanline Measurements Registered from an Object in Some Pose

Figure 7.4: Relationship between modular fixture configurations achieving simultaneous contact and scanline measurements

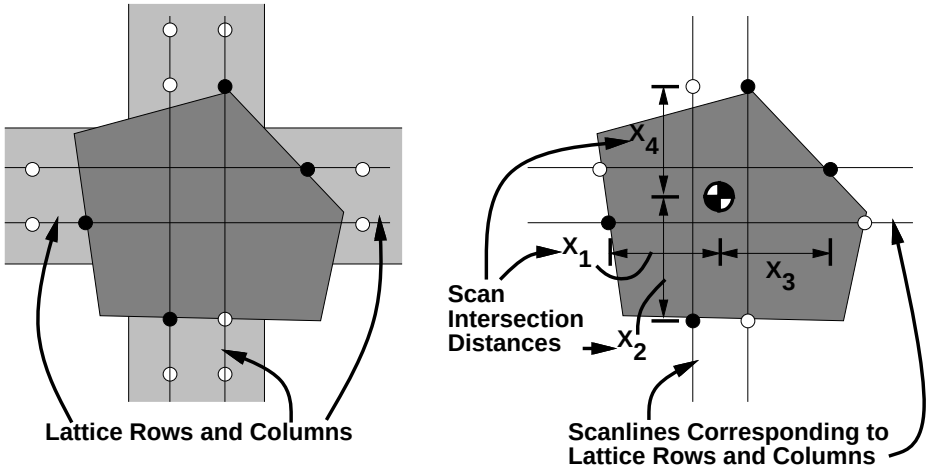


Figure 7.5: Relationship between modular fixture configurations achieving simultaneous contact and scanline measurements

and discretizing these coordinates to index a table entry containing the correspondence information [CJ91; LSW88; KSSS86; FMZ+91] (Figure 7.6).

These tables need to include table entries for every valid combination of discretized indexing coordinates and interpretations. By asserting that these discretized indexing coordinates correspond to integral pairwise differences, then every fixture configuration is included as an entry in the indexing table.

Consider a complete indexing table for the parallel-beam scanning sensor where the indexing coordinates correspond to the pairwise differences between scanline endpoints, where scanlines occur at integral multiples of λ_y $(k\lambda_y | k \in Int)$, and where the discretization is λ_x. Then the table must include separate entries for each configuration with different indexing coordinates. Each one of these indexing table entries characterizes a region in configuration space, of which the simultaneous contact configuration is a vertex.

Indexing tables are constructed by intersecting two types of configuration space curves, and then predicting indexing table entries at all of these intersections. The two types of curves are discretization boundary curves and correspondence boundary curves. Discretization boundary curves define configurations for which the indexing coordinates achieve integral multiples of the discretization $k\delta$. Correspondence boundary curves define configurations for which the correspondence interpretation changes. The basic premise of the indexing construction method is that these curves form an arrangement, and that within each cell in the arrangement, the discretized coordinates and correspondence interpretation

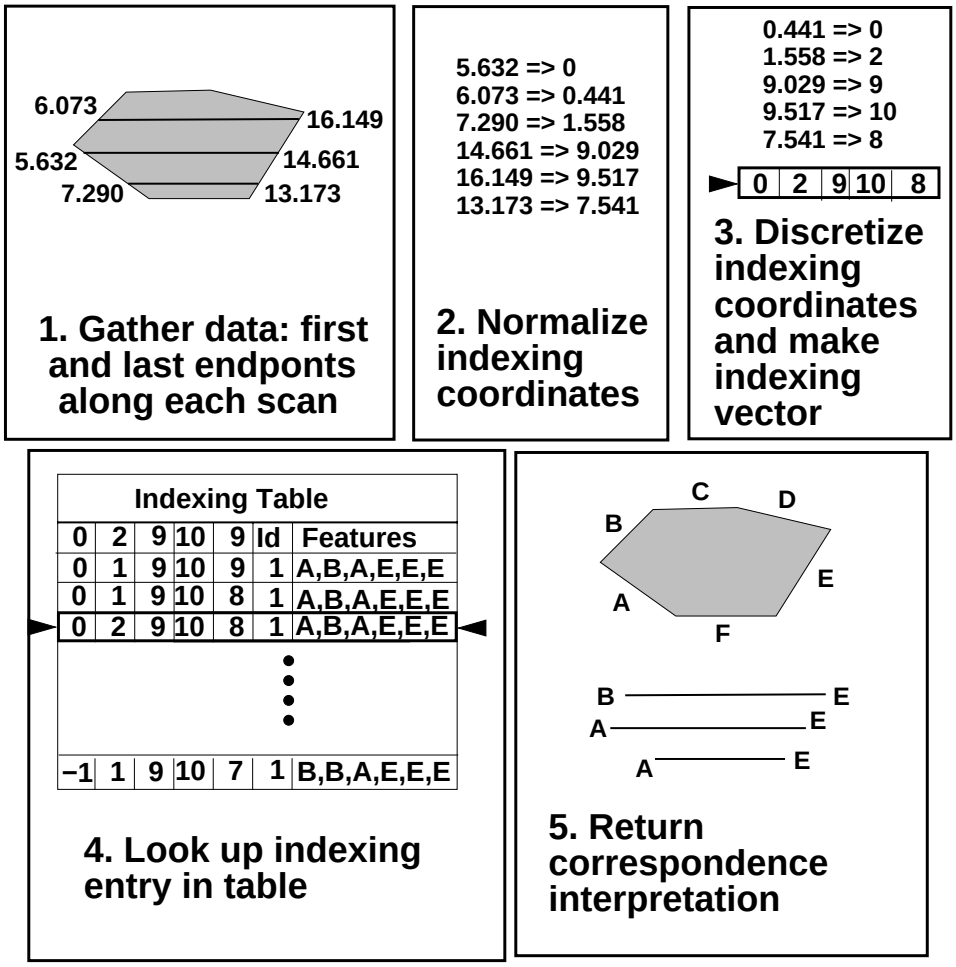


Figure 7.6: Indexing techniques involve discretizing the sensed data to index the entry containing valid interpretations

remain constant, and therefore require only a single indexing table entry. In indexing space, this process is depicted by Figure 7.7; the discretization hyperplanes partition indexing space into equivalence classes which discretized to the same the indexing coordinates, and the different colored patches characterize the correpondence interpretation of the configurations. It turns out that it is easier to enumerate the cells in the arrangement in configuration space by virtue of the lower dimension. Therefore, we project these curves down onto configuration space and enumerating the cells in the arrangement formed by these curves Figure 7.8.

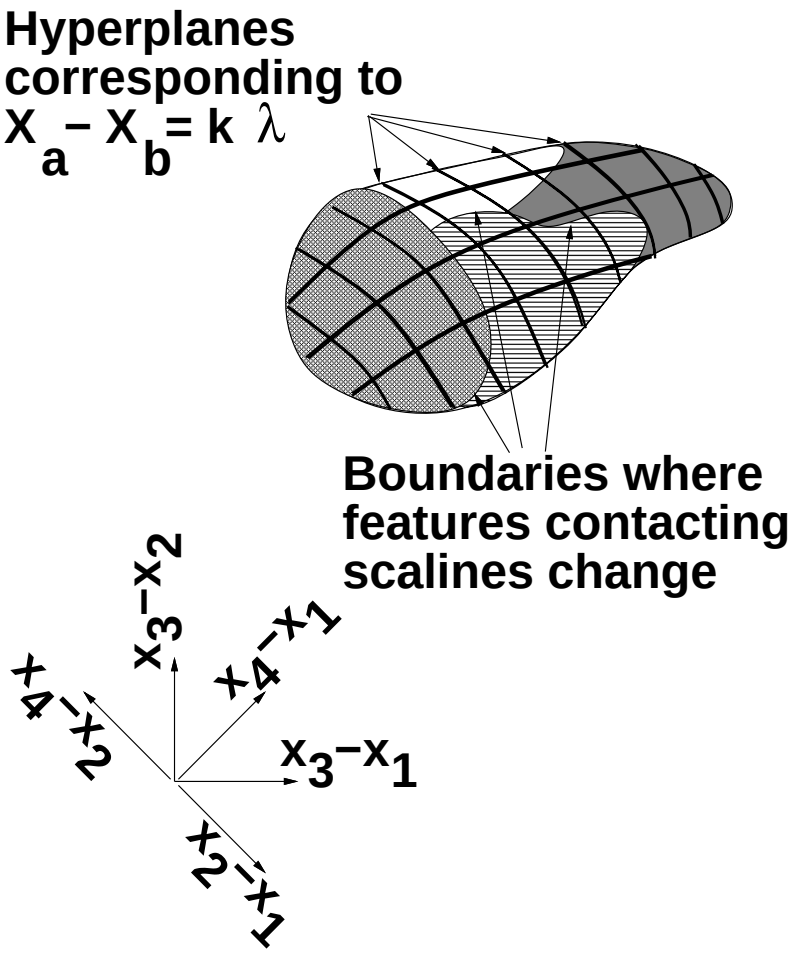


Figure 7.7: Consider all of the constraints in the larger indexing space: the constraints that the x difference between two intersections is a multiple of λ_x and the curves which signify the boundaries where the contact feature changes

We only need to modify this algorithm slightly to realize a generic modular fixture design algorithm, because the two problems are closely realted. For modular fixures, we

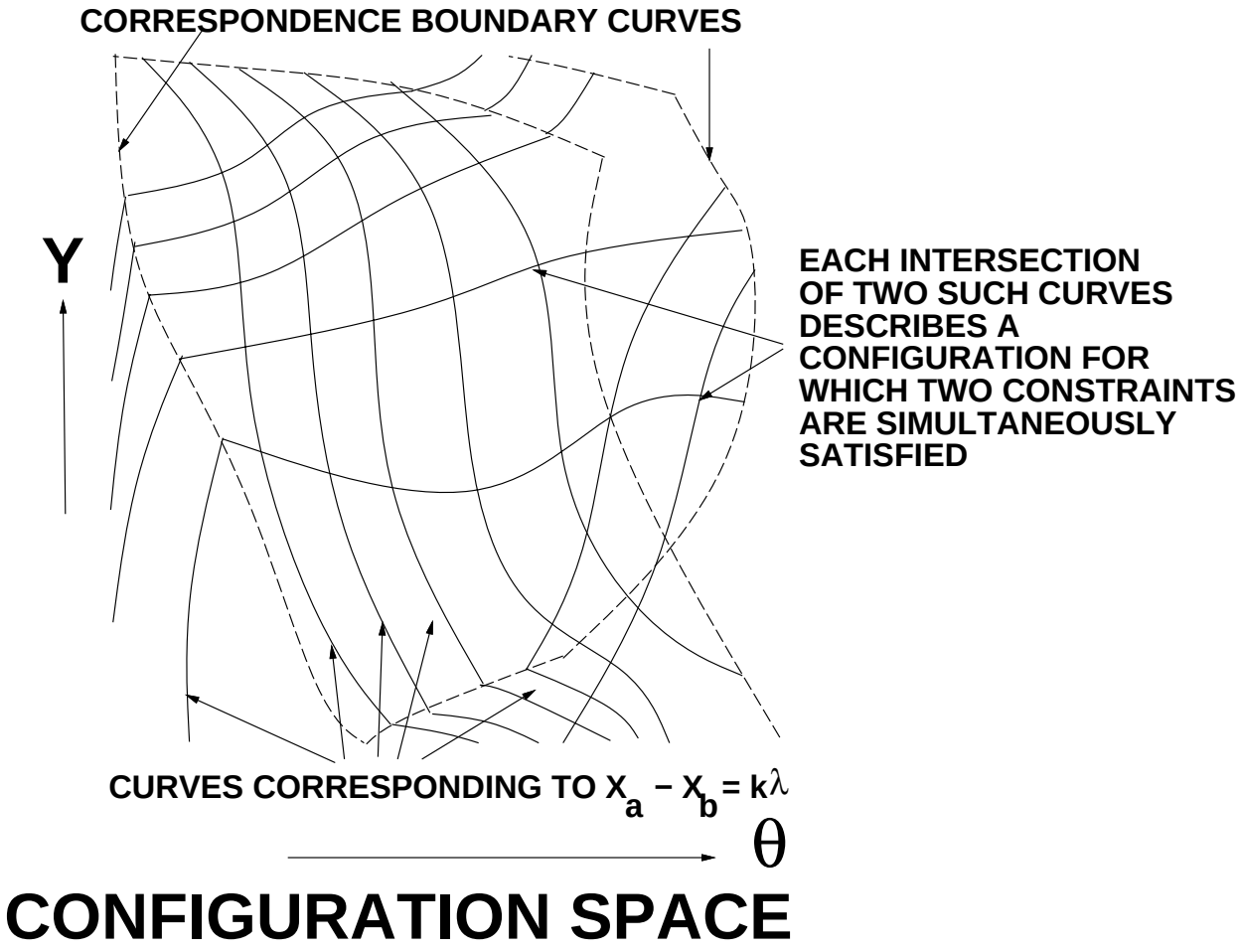


Figure 7.8: Complete indexing tables are constructed by a reduction to enumerating cells in the arrangement formed by two types of curves: discretization boundary curves and correspondence boundary curves. Fixture designs correspond only to intersections of discretization boundary curves.

are not concerned with enumerating table entries for each valid combination of discretized indexing coordinates and correspondence interpretations. Rather, we are only concerned with enumerating all intersections of l discretization boundary curves. Furthermore, we can use the correspondence boundary curves in Figure 7.8 to perform a sweep algorithm to efficiently enumerate these intersections. The sweep algorithm will enumerate configuration space intersections but the intersected curves will change according to the correspondence boundaries.

7.3.4 Advantages of Observing This Duality

The two main advantages of this duality it that it allows us to transfer knowledge from one domain to another, and that it provides us with a general, efficient, sweep algorithm approach for enumerating fixture designs. Transfering knowledge is important because it can highlight equivalence classes of fixtures. Consider the fixture vise system which is characterizable by four degrees of freedom (x, y, θ, σ) where (x, y, θ) characterize the pose of the workpiece and σ characterizes the articulated degree of freedom of the fixture; the dual scanning problem is characterizable by two degrees of freedom (y, θ) after normalizing away the x degree of freedom by setting one of the scanline endpoints to $x = 0$. Since the dual scanning system is characterizable by two degrees of freedom, then we can also characterize the modular fixturing system in terms of two degrees of freedom, and should also be able to identify two separate types of equivalence classes for fixtures; these correspond to translating the fixture up and down by the vise row spacing and translating the fixture left and right by the vise column spacing.

7.4 Novel Modular Fixture Toolkits

Given a general methodology for designing fixtures for arbitrary minimal fixture toolkits, the next step is to describe some of these novel minimal fixture toolkits which can use this algorithm. In this section, we describe three minimal modular fixture toolkits.

7.4.1 Three-Jaw Fixture Chuck

Figure 7.9 depicts a planar three jaw chuck modular fixturing toolkit. Pegs are inserted into the lattice holes inlaid in the fixture plates mounted on the jaws of a chuck,

and the workpiece is fixtured by tightening the chuck.

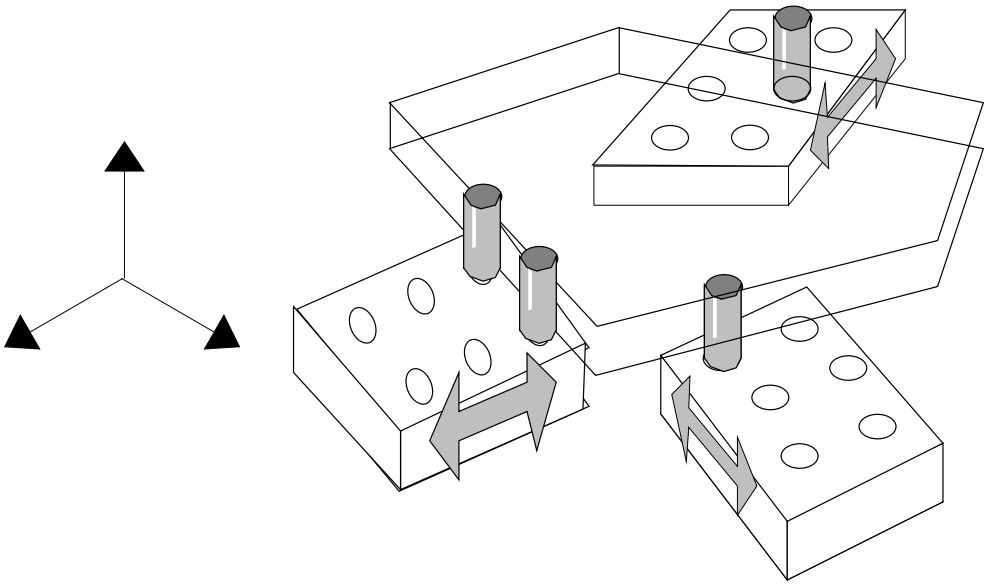


Figure 7.9: A three jaw chuck modular toolkit.

7.4.2 Four-Jaw Fixture Chuck

Figure 7.10 depicts a planar four jaw chuck modular fixturing toolkit. Pegs are inserted into the lattice holes inlaid in the fixture plates mounted on the jaws of a chuck, and the workpiece is fixtured by tightening the chuck.

7.4.3 Three Dimensional Tetrahedral Chuck

Figure 7.11 depicts a tetrahedral four jaw chuck modular fixturing toolkit. Pegs are inserted into the lattice holes inlaid in the fixture plates mounted on the jaws of a chuck, and the workpiece is fixtured by tightening the chuck.

7.5 Heuristics to Guide the Enumeration Routine

Even though the computation only involves processing quartets of model features, models with hundreds of features may take many hours on conventional computers. Furthermore, performance will surely decrease for toolkits with less structure than was found

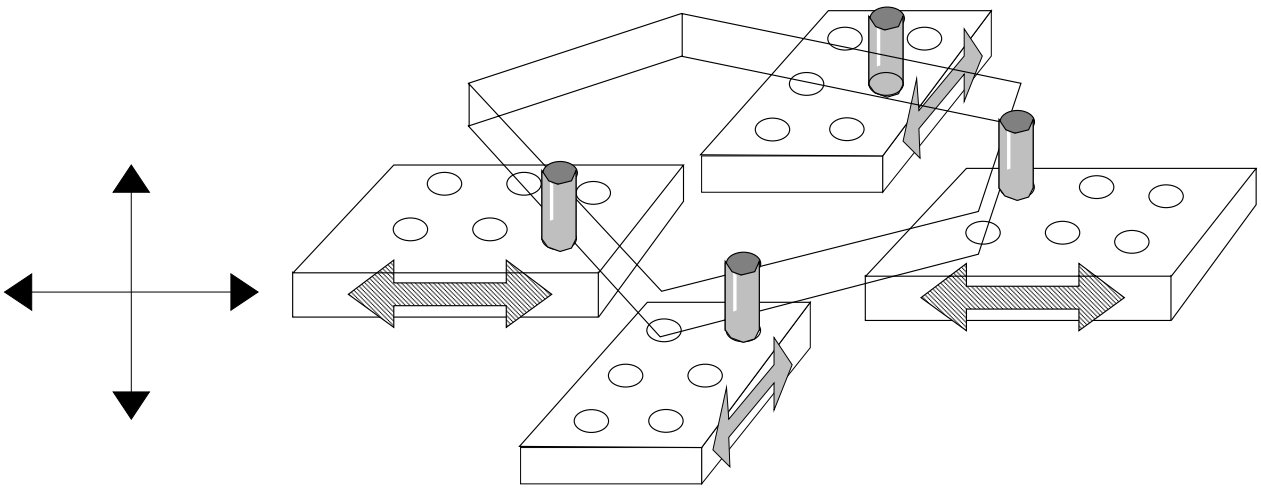


Figure 7.10: A four jaw chuck modular toolkit.

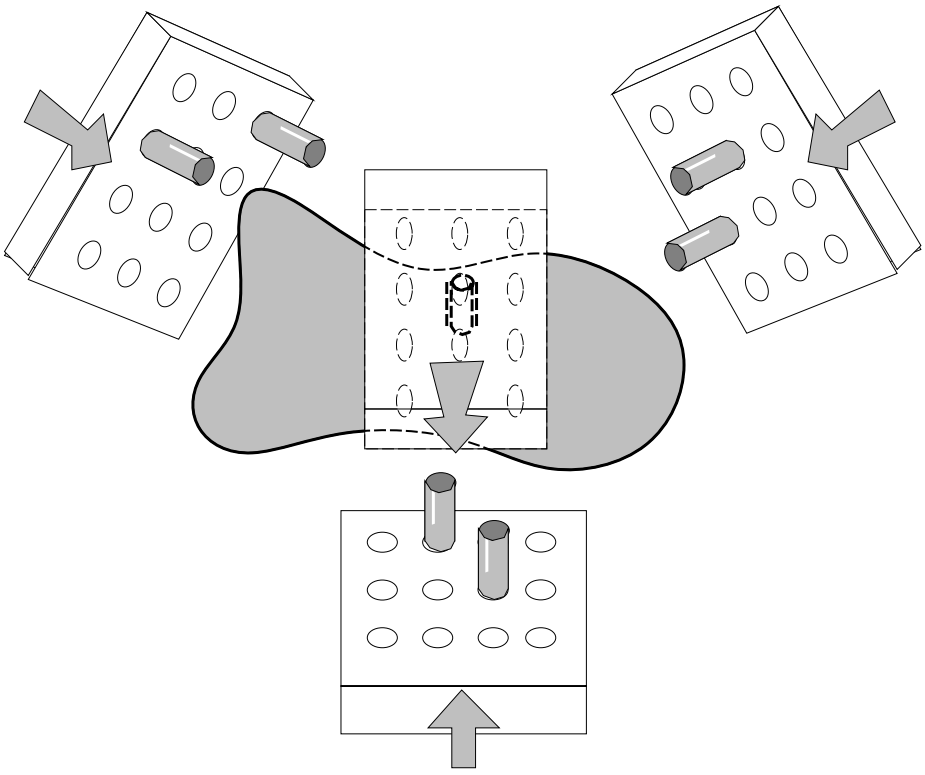


Figure 7.11: A three dimensional four jaw chuck modular toolkit.

in the translating clamp toolkit and fixture vise toolkit, and performance issues will become more critical for three dimensional fixture toolkits (tetrahedral chuck), with seven contacts and seven degrees of freedom.

In many applications, a single valid fixture suffices, and therefore, complete enumeration strategies are suboptimal. To improve performance to satisfy customer requirements, heuristics will be used to guide the search; in this section, we present three such heuristics: a metric for ranking feature tuples for fixture quality, a metric for ranking the probability that a feature tuple will admit a simultaneous contact configuration with some peg configuration, and a hill climbing heuristic for solving discrete systems such as finding peg configurations.

7.5.1 Ranking Feature Tuples

In this section, we present two metrics for evaluating the probability that a particular tuple of features will admit a minimal modular fixture design. The robotics literature includes many metrics for comparing grasps [FC92; MC94]. The first metric is the largest ellipsoid enclosed in the wrench space polytope corresponding to unit forces applied at each of the feature vertices, and the second metric is the product of the surface areas of all of the features. The largest enclosed ellipsoid provides us with an estimate of the ranges of wrenches we can resist with maximum leverage, and this provides some estimate of the quality of the corresponding fixture. The second metric, the product of the surface areas of the feature tuples, characterizes provides us with some estimate concerning the probability that we can find a peg configuration which achieves simultaneous contact with the feature tuple. This metric can be adapted to correspond to the area of the associated hypersurface patches in indexing space; *i.e.,* for the fixture vise, the metric could correspond to the sum of the products of the areas of the two pairs of left and right features.

7.5.2 Discrete Hill Climbing Heuristic

In this section, we propose a discrete hill climbing heuristic for efficaciously finding a valid fixture; *i.e.*, given a feature tuple, finding a peg configuration which achieves simultaneous contact and exhibits force closure. The approach begins with a random peg configuration, and proceeds by shifting one peg to a neighboring location until it achieves a peg configuration which admits a force closure configuration. To guide the hill climbing, *i.e.*

to choose which peg to shift, this approach relies on a metric which estimates the distance from the current peg configuration to a valid force closure configuration.

We propose a metric which characterizes the discrepancy between the current peg configuration and a valid configuration as the scaled sum of a two errors $E_{sc} + \lambda\, E_{fc}$: E_{sc}, an error component corresponding to the lack of achieving simultaneous contact, and E_{fc}, an error component corresponding to the lack of achieving force closure. As a function of the object/fixture configuration (x, y, θ, σ), define error functions $E_{sc}(x, y, \theta, \sigma)$ $E_{fc}(x, y, \theta, \sigma)$ which characterize the respective errors. Then, the total error is global minimum of these errors over all configurations: $\min_{x,y,\theta,\sigma}\ E_{sc}(x, y, \theta, \sigma) + \lambda\, E_{fc}(x, y, \theta, \sigma)$.

The minimum sum squared error between the pegs and the transformed features is a reasonable error function $E_{sc}(x, y, \theta, \sigma)$. We can compute the minimum sum squared error between four feature segments and four pegs algebraically in constant time by noticing there are only 3^4 different cases. The minimum distance between a peg and a feature segment can correspond to the distance between a peg and the extended feature, which can be defined algebraically, or the distance between a peg and a vertex, which also be defined algebraically (Figure 7.12). Therefore, we can solve for the configuration $(x, y, \theta, \sigma)_{min}$ which globally minimizes the error function assuming each particular case (similar to the localization method described in Chapter 4), and computing the global minimum by checking all 3^4 cases.

Many researchers have been suggested metrics to evaluate grasp quality [LS88; MP89a; BYT90; FC92; CCB89; BG94] which we can use as error functions $E_{fc}(x, y, \theta, \sigma)$. The difficulty lies in characterizing these error functions as rational algebraic functions.

We do not have any performance measurements since we have not yet implemented these discrete hill-climbing heuristics. Hill-climbing strategies may fail to find a valid fixture because they get stuck in local minimum. One method to overcome problems associated with local minima is to vary the λ parameter as a function of time, *i.e.*, simulated annealing.

7.6 Discussion

Although minimal fixture toolkits can generically fixture arbitrary workpieces, we believe that such toolkits should be equipped with two additional degrees of freedom to enable these fixtures to be used in conjuntion with four degree of freedom manipulators. Planar modular fixture toolkits are only capable of fixturing prismatic workpieces, and if

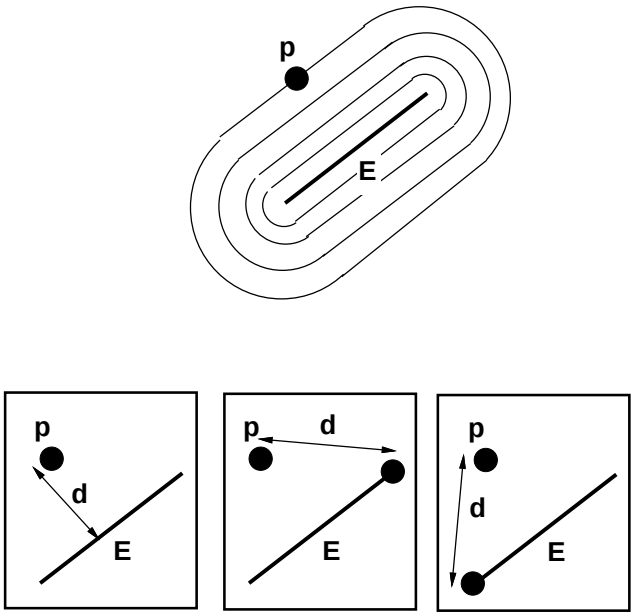


Figure 7.12: The minimum distance between a peg and a feature segment can correspond to the distance between a peg and the extended feature, or the distance between a peg and a vertex

the particular feature requires a non-vertical operation, then the fixture cannot be used in conjuntion with a SCARA four degree of freedom manipulator. Even three dimensional tetrahedral chuck fixtures only achieve a finite number of simultaneous contact configurations, and therefore, do not generically accomodate vertical operations. Therefore, we need to augment minimal fixture toolkits with two rotational degrees of freedom in order to achieve a desired orientation which admits vertical insertions and operations.

7.6.1 Conclusion

In this chapter, we presented a complete algorithm for designing fixtures for arbitrary objects using minimal toolkits, where the term minimal refers to a fixture where the total number of degrees of freedom is equal to the number of contacts. In this case, there are only a finite number of configurations which can achieve force closure, assuming a frictionless point contact model. This general algorithm stems from a duality we observed between modular fixturing and scanning sensing; it turns out that every modular fixture design corresponds to a scanning sensing configuration, and we can therefore adapt an algorithm for constructing complete indexing tables to a generic algorithm for enumerating

valid modular fixture designs for arbitrary objects.

Chapter 8

Constructing Complete Indexing Tables to Recognize Polyhedral Objects

Abstract

Model-based object recognition is an important task in machine vision, and most approaches are feature-based in that they extract interesting features from the image data, and match these features to model features. One of the most difficult subproblems in recognizing objects is the correspondence problem, the problem of determining which model features correspond to the sensed features. Indexing is one approach to the correspondence problem in which indexing coordinates are distilled from sensed feature groups, the indexing coordinates are quantized, and the quantized coordinates index a table entry containing the valid interpretations of the sensed feature group as model feature groups. The advantage of indexing is that a single table lookup efficiently compares the sensed feature group to all the model feature groups simultaneously. Indexing tables are precomputed offline and the technique assumes that experimentally measured indexing coordinates will discretize to the same indexing table entry as the predicted indexing coordinates. Indexing also relies on the assumption that the table contains entries for every valid combination of indexing coordinates and interpretations.

There are two major reasons the completeness issue has been overlooked: dense

sensing strategies and invariants. For dense sensing strategies, where each modeled object provides multiple sensed feature groups, if the table only includes one valid indexing table entry for all the object's sensed feature groups, indexing will still perform satisfactorily. Other indexing techniques have focused on systems which exhibit invariants, where the term invariant refers to a property which remains constant irrespective of the object or camera pose. Invariants enable model feature groups to be characterized by a single indexing cooridnate, and, thereby, a single indexing table entry.

We have developed a method for constructing complete indexing tables and this broadens the applicability of indexing to non-dense sensing systems which do not exhibit invariance. Using this technique, we implemented a recognition system which identifies polyhedral objects from unscaled orthographic projective image data. The sensed feature groups consist of pairs of rays (pairs of edges which share a coincident vertex); five indexing coordinates are extracted from each pair of rays: four orientation measurements and one length measurement. These coordinates are quantized to achieve integral indices for table lookup. In this chapter, we describe our technique for constructing complete indexing tables and present sizes of complete indexing tables for various objects and table discrtizations.

8.1 Introduction

Current machine vision systems are capable of comprehending and interpreting unconstrained scenes, and have become widely used in areas such as navigation, image processing, and image filtering. Currently available systems can comprehend generic scenes *given an unlimited amount of processing time and computational resources*, but these systems are too slow and resource expensive for many more applications such as identifying unknown objects from unconstrained image data. Reasonable performance usually involves constraining the problem in some way, such as requiring object models.

In Chapters 2 and 3, we described a method for constructing complete indexing tables for sparse sensing systems, where each experiment only provides a handful of measurements. In this chapter, we extend this methodology to dense sensing systems, where each experiment provides measurements from various model feature groups, such as machine vision. As we mentioned in previous chapters, the main drawback of indexing is that the indexing tables need to be complete; *i.e.*, the table must contain an entry for every interpretation and every indexing coordinate. Unfortunately, until recently, the issue of

indexing table completeness has been overlooked, and no method for constructing complete indexing tables has been presented.

The completeness issue has been igrnored for two main reasons: dense sensing strategies and invariants. Dense sensing strategies, such as identifying objects from image data, provide a large number of sense feature groups corresponding to a single object, requiring only a single correct indexing table entry to achieve correct operation. Secondly, many previous indexing techniques have focused on systems which exhibit invariants. The term invariant refers to a property which remains constant irrespective of the object and camera position. For systems which exhibit invariance, we can choose invariant indexing coordinates so that each model feature group can be characterized by a single indexing table entry. The literature includes many successful systems which combined indexing with invariant properties, Kalvin *et al.*, Schwartz and Sharir, and Forsyth *et al.* [KSSS86; SS87; FMZ+91]. Unfortunately, without a provably correct method for constructing complete indexing tables, indexing techniques are inapplicable to a multitude of tasks. For example there are no geometric invariants for three-dimensional point sets for orthographic [CJ91] or perspective projection [BWR93].

Clemens and Jacobs developed systems for identifying three dimensional models from image data using non-invariant indexing [CJ91]. In their systems, indexing coordinates were extracted from groups of image points, and these coordinates were used to look up the corresponding groups of model points. Since they relied on indexing generic point groups, for which two rotational degrees of freedom cannot be normalized away, their approach involved enumerating indexing table entries for two dimensional stratifications in indexing space. They enumerated the indexing table entries by uniformly sampling the viewing directions because "we do not know how to analytically determine exactly which parameterized buckets intersect the 2-D surface that a model may produce even in the absence of error"[CJ91, page 1012]. Constructing indexing tables via sampling may result in incomplete tables causing false negatives, *i.e.,* sensed feature groups which are not identified as model feature groups because the corresponding indexing table entry is missing.

In this chapter, we present a method for guaranteedly constructing complete indexing tables. The main idea is that each indexing table entry corresponds to a finite number of continuous regions (cells) in configuration space, and cells are separated by only two classes of boundaries: discretization boundaries separating configurations which discretize to different indexing coordinates, and correspondence boundaries which bound the configu-

rations for which a model feature group is visible. Our method enumerates all of the entries in the indexing table by enumerating all of the cells in the arrangement in configuration space.

It is also interesting to notice that our arrangement based approach is reminiscent of work done on constructing complete aspect graphs. Aspect graphs [GCS91] qualitatively represent images in such a way that entire regions share the same aspect (the definition of aspect is application and task dependent). For various viewing directions, a single object can be consistent with multiple aspects. Gigus *et al.* [GCS91] constructed complete aspect graphs by enumerating all of the events for which the aspect possibly changed. Each such event corresponded to a locus of configurations, and configurations which were all on the same side of all of the events had the same aspect graphs. Thereby, the problem of constructing complete aspect graphs also reduces to the problem of enumerating cells in the arrangement (defined by the event loci).

In order to explain the construction methodology, we step through the process of constructing complete indexing tables for recognizing polyhedral objects from ray pairs assuming unscaled orthographic projective image data. This task is similar to the problem discussed by Thompson and Mundy [TM87], and the same task will also be discussed in Chapter 8.

8.1.1 Outline

In this chapter, we describe an indexing technique which utilizes our complete indexing construction methodology to recognize polyhedral objects from image data assuming unscaled orthographic projective image data. In section two, we define the task in terms of extracting indexing coordinates from sensed feature groups consisting of image ray pairs. In section three, we discuss our complete indexing construction methodology and define configuration space curves representing indexing coordinate boundaries and correspondence boundaries. Our implementation utilizes recently developed algebraic and numerical techniques for intersecting these curves, and these methods and an example are presented in section four. In section five, we present the sizes of complete indexing tables for various objects and discretizations. We conclude by highlighting the results.

8.2 Task

In this chapter, we utilize an indexing technique to recognize modeled polyhedral objects from image data by extracting features (rays) from the image and using indexing techniques to solve the correspondence problem; *i.e.*, to enumerate the consistent interpretations of sensed features (rays) as model features. We mainly focus on the construction of complete indexing tables.

8.2.1 Definitions

Definition 8.1 *Feature* refers to a characteristic of a modeled object which can be detected from sensor data, and the term feature group refers to a set of one or more such features. For example, the pyramid polyhedron shown in Figure 8.1 contains 18 ray features, where each ray corresponds to a pair of coincident edges. The correspondence problem can be defined as providing a map from sensed feature groups to valid model feature groups interpretations.

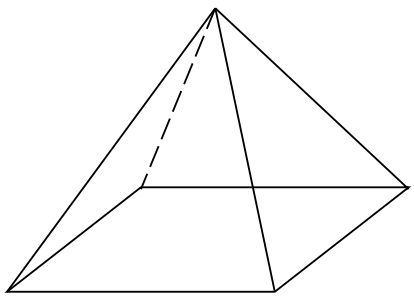


Figure 8.1: Pyramidal polyhedral object

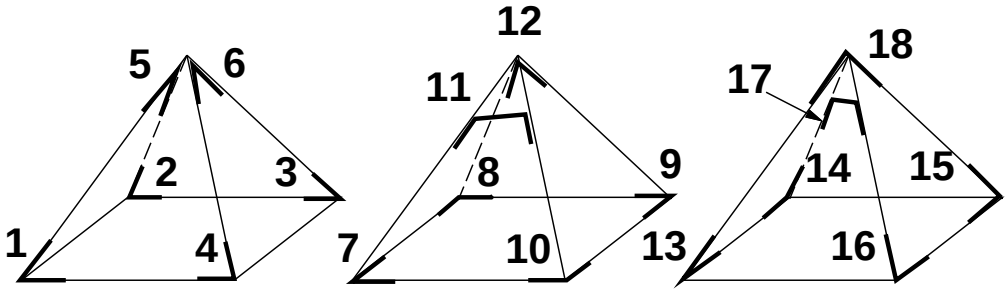


Figure 8.2: The 18 ray features included in the pyramid polyhedron

Definition 8.2 *Configuration* $\vec{x}$ characterizes the observable state. In this case, configuration $\vec{x}$ corresponds to the object's pose with respect to the camera. *Configuration space* (C) describes the set of all configurations ($\vec{x} \in$ C). Under unscaled three-dimensional orthographic projection, configuration space can be defined by the parameters x, y, θ, ϕ, ψ; z is not a configuration variable because it is unobservable. Furthermore, the process of extracting indexing coordinates can normalize away additional degrees of freedom. In this technique, extracting relative orientations and length measurements normalizes away rotation and translation in the image plane, leaving only the out of plane rotations.

Definition 8.3 *Indexing coordinate* y_i refers to a real valued scalar data value distilled from a group of sensed features; $\hat{y}$ refers to a set of n simultaneously observed indexing coordinates. *Indexing space* ($\mathbb{I}$) describes the set of all possibly observed vectors.

Definition 8.4 *Prediction map function* M_f for a given model feature group f, maps from configuration space into indexing space: $\hat{M_f} : \mathrm{C} \rightarrow \mathbb{I}$.

Definition 8.5 *Indexing table* inverts the interpretation mapping function $\cup_f M_f^{-1}$ *from* the indexing coordinates $\hat{y}$ *to* the model feature group f. $T_{\bigcup_f}(\hat{y}) = \{< f > \mid \exists \vec{x} s.t. \vec{M_f}(\vec{x}) = \hat{y}\}$

The predicted indexing coordinates for each model feature group is a function of the object's configuration. Viewed in indexing space, the set of predicted indexing coordinates consistent with each model feature group forms a manifold; the correspondence problem can be defined as: *determine the (model feature group) manifold(s) nearby a given indexing vector.*

8.2.2 Indexing Coordinates for Model Feature Groups

Basically, indexing techniques solve the correspondence problem by extracting sensed features from the image data (rays), enumerating sensed feature groups (pairs of rays), and then efficiently enumerate consistent interpretations (model feature groups) via the indexing table. For our recognition system, indexing is performed as shown in Figure 8.3. First, we extract rays (model features) from the image, and then we enumerate sensed feature groups consisting of pairwise combinations of sensed features; next, we distill indexing coordinates from the group of sensed features and discretize the indexing coordinates to index into a table entry, where the table entry contains the correspondence information.

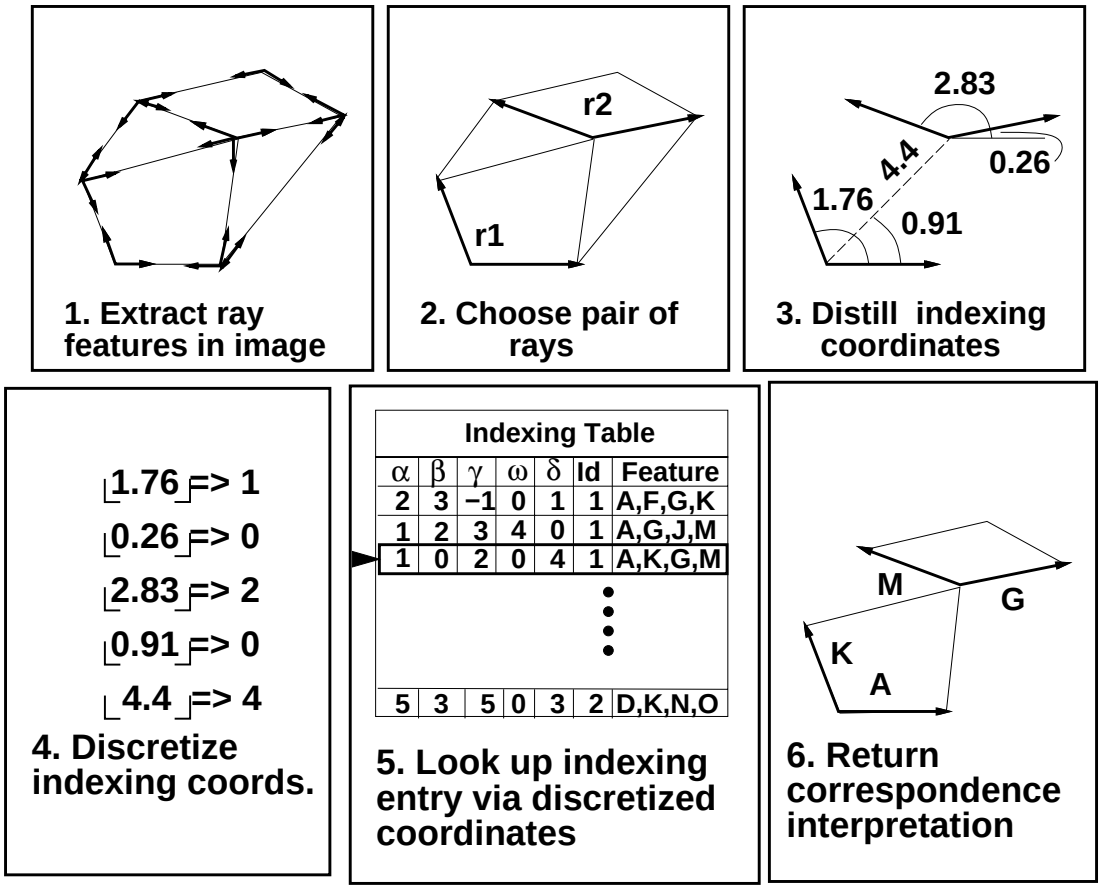


Figure 8.3: The indexing technique

Our indexing technique distills four relative orientation measurements and one length measurement from each pair of rays (r_1, r_2). This distillation procedure implicitly normalizes away three of the five degrees of freedom of unscaled orthographic projection: x, y, R_z, rotation and translation in the image plane. Thereby, only two (rotational) degrees of freedom remain, the rotations out of the image plane.

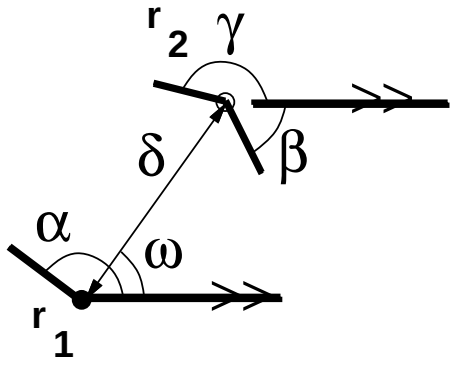


Figure 8.4: Indexing coordinates $(\gamma, \alpha, \beta, \omega, \delta)$ are extracted from each pair of rays

8.3 Complete Construction Method

In this chapter, we describe an approach for constructing complete indexing tables. The approach enumerates indexing table entries by enumerating cells in an arrangement in configuration space. The first observation is that discretizing indexing coordinates are to achieve integral indices corresponds to partioning the indexing space into equivalence classes of hypercubes consisting of indexing points which all map to the same discretized indexing coordinates. For this reason, all of the indexing table entries for a model correspond to discretization hypercubes which cover the model stratification; an indexing table, then, can be alternatively viewed as a regular cell covering of the model manifolds. The second observation is that it is easier to enumerate the regular cell coverings of the model manifolds in configuration space than indexing space because of the lower dimension.

There are only two types of boundaries which delimit indexing table entries: discretization boundaries which separate cells with different quantized indexing coordinates, and correspondence boundaries which delimit the configurations for which a model feature group is visible. After projecting these discretization boundaries and correspondence bound-

aries down onto configuration space, constructing complete indexing tables is analogous to enumerating cells in the arrangement formed by these curves.

8.3.1 Configuration Space

In this particular system (unscaled orthographic projection), there are only two degrees of freedom. Since the process of distilling the indexing coordinates normalizes away translation and rotation in the image plane, the object's configuration can be parameterized by two variables $(a, b) \in \mathbb{R}^2$. Figure 8.5 shows the pyramid object corresponding to various configurations.

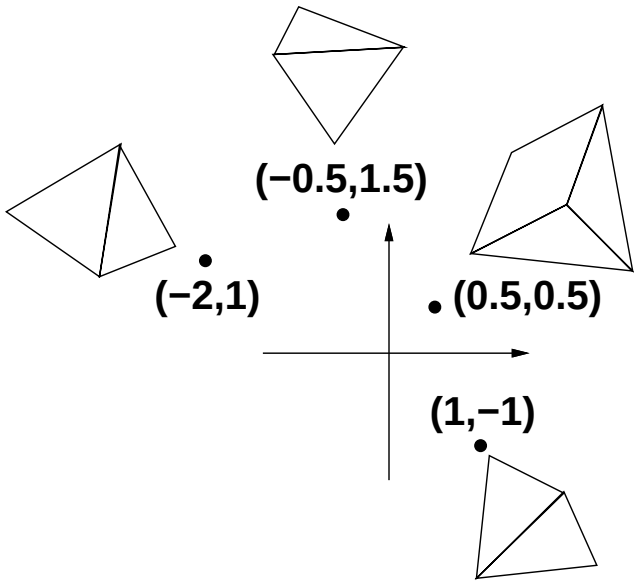


Figure 8.5: Rotating the pyramid by various configurations

8.3.2 Main Idea

The main idea is that we want to enumerate every valid indexing table entry by enumerating all of the different discretized indexing coordinates and all of the interpretations consistent with each coordinate. Our approach is based upon the simple observation that two different configurations will correspond to two different indexing table entries if and only if the two configurations are separated by either discretization boundaries or correspondence boundaries. Discretization boundaries characterize the configurations which straddle two

different quantizations: in other words, if indexing is separated on unit boundaries, *i.e.*, $\langle 2, 3\rangle \rightarrow 2$ and $\langle 3, 4\rangle \rightarrow 3$, then the value 3 straddles two ranges. Correspondence boundaries characterize configurations at which each model feature group ceases to be visible.

Although this method of constructing complete indexing tables has already been presented for sparse sensing strategies [WC95a] in which only a single feature group is observed in each experiment, there remain some subtle differences which must be dealt with to handle dense sensing strategies, and that is why we will describe the approach in detail.

Generic machine vision applications involve dense sensing strategies in which multiple feature groups are observed from each experiment. One key distinction between sparse sensing strategies and dense sensing strategies is that for dense sensing strategies, multiple feature groups can be observed in each experiment; therefore, we want to construct separate complete indexing tables for each and every model feature group, and define a separate configuration space for each model feature group (Figure 8.8).

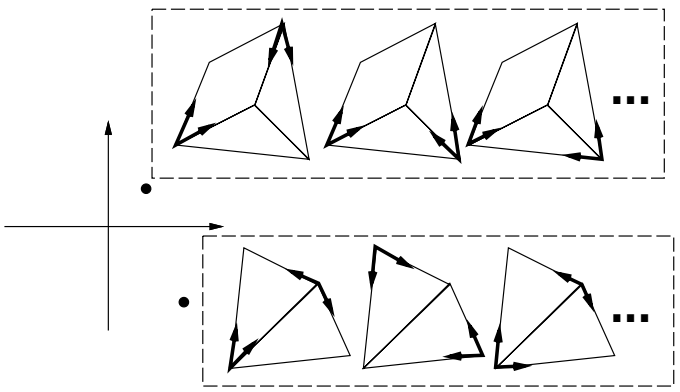


Figure 8.6: Each image contains multiple sensed feature groups

We define a separate indexing space for each model feature group, so that each configuration corresponds to a specific feature group. We construct discretization boundaries and correspondence boundaries for each configuration space, and then we enumerate complete indexing tables by enumerating the cells in each arrangement for each configuration space (Figure 8.7).

The basic idea is that, for each configuration space (model feature group), each indexing table entry corresponds to a finite number of continuous regions (cells) in configuration space, where the cells are separated by only two classes of boundaries: discretization

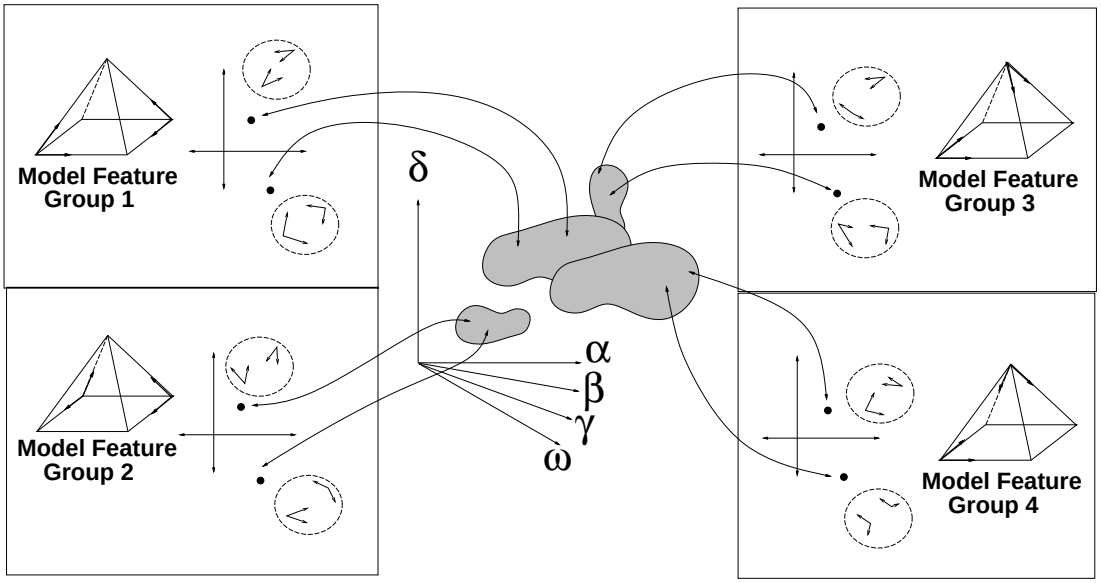


Figure 8.7: All of the predicted indexing coordinates correspond to at least one of the mappings and configuration spaces for a model feature group.

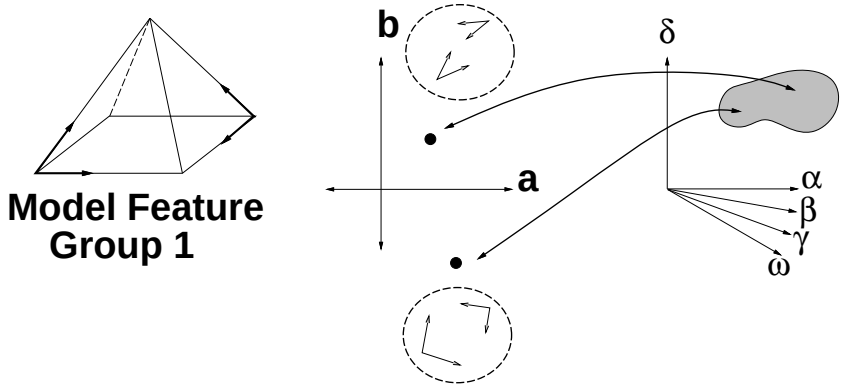


Figure 8.8: For each model feature group, the coordinates in the five dimensional indexing space correspond to (a, b) configurations.

boundaries separating configurations which are discretized to different indexing coordinates, and correspondence boundaries characterizing configurations at which each model feature group ceases being visible (Figure 8.9). Furthermore, we only need to enumerate a single witness in each of the different cells, because if we predict an indexing table entry for at least one witness in each cell, we are guaranteed to construct a complete indexing table.

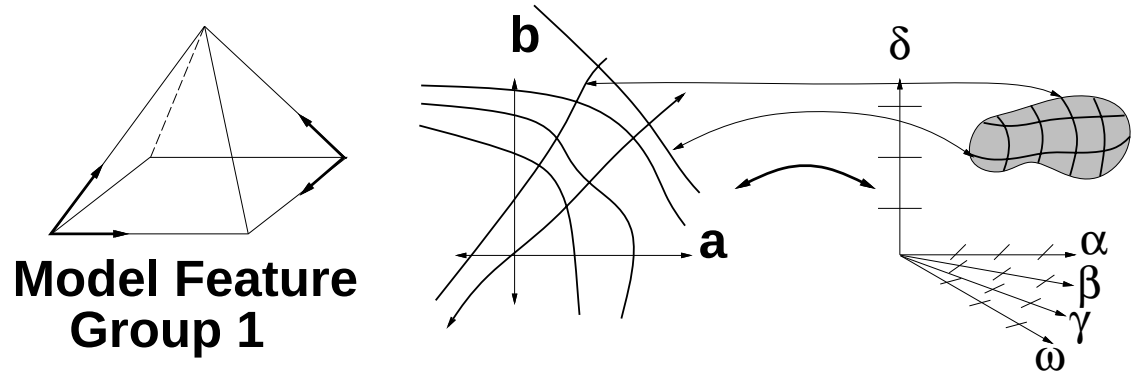


Figure 8.9: For each model feature group (pair of model rays), the discretization boundaries corresponding to partitioning indexing space into equivalence class discretization hypercubes are characterized by configuration space curves of codimension one

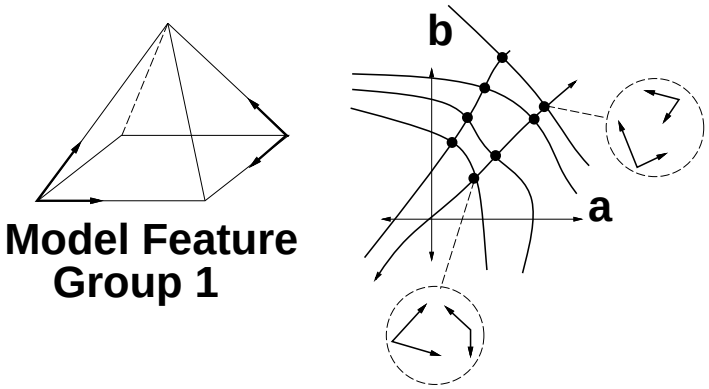


Figure 8.10: By enumerating a witness (vertex) for each cell, and predicting indexing table entries for each such witness, we can guaranteedly construct a complete indexing table. Witnesses can correspond to every intersection of k boundary curves in a k dimensional configuration space.

This technique involves three reductions: first, we defined separate configuration spaces for each model feature group; second, we reduced the problem of enumerating indexing table entries to the problem of constructing a complete cell covering of the indexing

manifold in indexing space; and third, we reduced the problem of constructing a complete cell to the problem of enumerating cells in an arrangement in configuration space, defined by curves which are the projections of the cell boundaries in indexing space down into configuration space. Finally, indexing tables can be constructed by predicting the indexing coordinate entries at each intersection of k boundary curves in a k dimensional configuration space.

8.3.3 Notation

We adopt the convention that capital letters refer to points which have been rotated corresponding to the quaternion $(1, a, b, 0)$. For the purposes of writing algebraic expressions for the curves, we will use five different variables: the positions S, P and the direction vectors E, E' (Figure 8.11). S refers to the start point of edge E. We also use the projection operation Π to characterize orthographic projection: $\Pi(x, y, z, p) \rightarrow (x, y, 0, p)$.

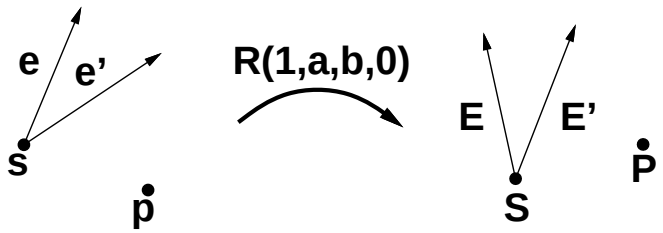


Figure 8.11: Points and features transformed by rotation $(R(1, a, b, 0))$ will be denoted bt capital letters. We will define the mathematical expressions in terms of the points S, P and the direction vectors E, E'

8.3.4 Correspondence Boundary Curves

Correspondence boundary curves describe configurations for which the model feature group ceases being visible. For our recognition system, becoming invisible corresponds to a vertex becoming occluded by another edge which is *in front* of the vertex. For simplicity, we express the visibility criterion in terms of an image point being colinear with an extended edge.

$$(\Pi(P) \Leftrightarrow \Pi(S)) \times \Pi(E) = 0 \tag{8.1}$$

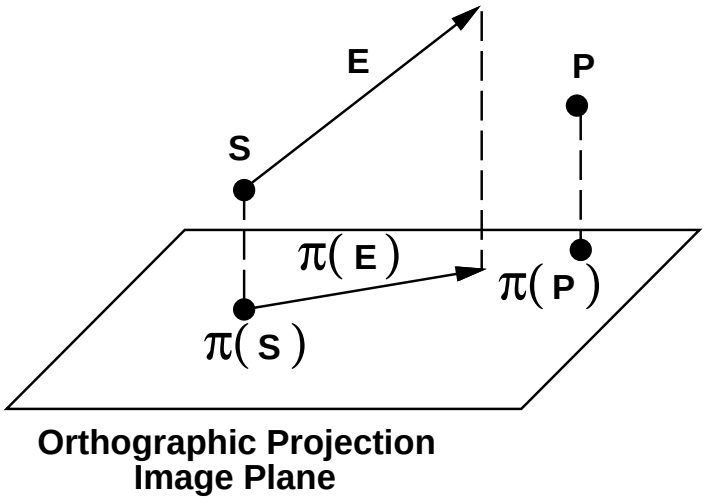


Figure 8.12: Correspondence curves are characterized by the image points being colinear with other extended edges.

8.3.5 Discretization Boundary Curves

In this section, we define discretization boundary curves for our recognition system; discretization boundary curves are characterized by expressions in terms of configuration space variables. Since our indexing coordinates include both relative orientation and length measurements, we need to define two different classes of discretization boundary curves.

Orientation Between Two Projected Edges

First we define discretization boundary curves corresponding to relative orientations. In this section, we express the constraint that the orientation between two projected vectors achieves a desired value. Although we could compute this in a number of ways, we chose a characterization which involved a ratio between the dot product and cross product because it does not require normalizing the edges to achieve unit length. This technique relies upon intersecting correspondence and discretization boundary curves, and renormalization would needlessly complicate the discretization boundary curve expressions.

$$(\Pi(E) \cdot \Pi(E')) - \tan(\theta)(\Pi(E) \times \Pi(E')) = 0 \tag{8.2}$$

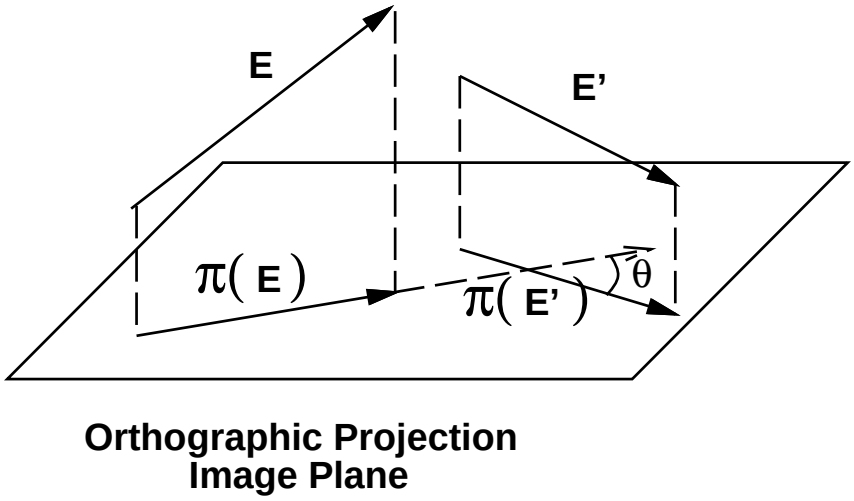


Figure 8.13: Orientation-DBC curves characterize the configurations for which the angle between the projections of two edges achieves a specified value.

8.3.6 Length Between Two Projected Points

Next we define discretization boundary curves corresponding to length measurements. In this section, we express the constraint that the distance between two projected points achieves a desired value.

$$\Pi(E) \cdot \Pi(E) \Leftrightarrow \delta^2(1 + a^2 + b^2)^2 = 0 \tag{8.3}$$

8.4 Theoretical Framework

In this section, we discuss parameterizations, algebraic, and numerical techniques used to intersect correspondence and discretization boundary curves. First, we describe our quaternion-based parameterization representing the set of out-of-plane rotations. Then we discuss algebraic techniques for intersecting curves. Next, we present an algebraic elimination method (Dixon resultants) for intersecting curves, and step through an example.

8.4.1 Parameterizing Configuration Space Via Quaternions

Recall from section 8.2.2, that the indexing coordinates implicitly normalize away all but the out-of-plane rotational degrees of freedom. Quaternions are a classical mathematical tool for representing arbitrary three-dimensional rotations by four dimensional

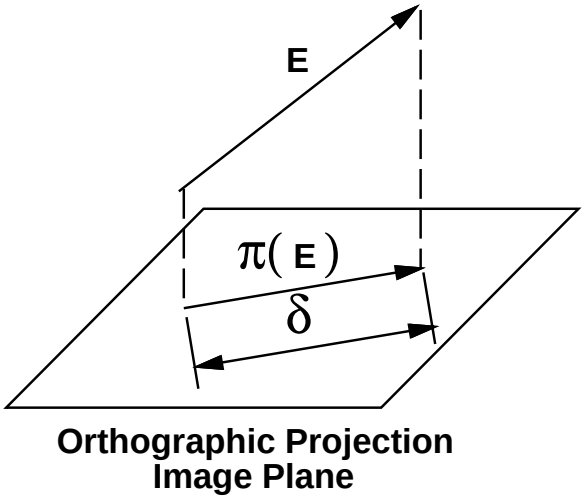


Figure 8.14: Length-DBC curves characterize the configurations for which the length of a projection of an edge achieves a specified value.

complex numbers. Any rotation in $SO(3)$ can be characterized by a rotation around the axis ω through θ radians, where the quaternion parameterization (q_0, q_1, q_2, q_3) corresponds to $(\cos(\frac{\theta}{2}), \omega_x \sin(\frac{\theta}{2}), \omega_y \sin(\frac{\theta}{2}), \omega_z \sin(\frac{\theta}{2}))$.

In order to characterize only the out-of-plane rotations, we utilize constrained quaternions $(q_0 = 1, q_1 = a, q_2 = b, q_3 = 0)$ where the first component q_1 is always one and the fourth component q_3 is always zero; constrained quaternions represent the set of rotations such that planar rotations (in the x, y plane) are irrelevant. The drawback of this constrained quaternion parameterization is that it produces irrelevant rotations in the x, y plane, but these effects are easily ignorable.

In order to utilize quaternions in a matrix representation, we want to define a rotation matrix $R(1, a, b, 0)$ in homogeneous coordinates corresponding to quaternion $(q_0 = 1, q_1 = a, q_2 = b, q_3 = 0)$ is given in equation (8.4).

$$R(1,a,b,0) = \begin{bmatrix} a^2 + 1 \Leftrightarrow b^2 & 2ab & 2b & 0 \\ 2ab & b^2 + 1 \Leftrightarrow a^2 & \Leftrightarrow 2a & 0 \\ \Leftrightarrow 2b & 2a & 1 \Leftrightarrow a^2 \Leftrightarrow b^2 & 0 \\ 0 & 0 & 0 & 1 + a^2 + b^2 \end{bmatrix} \tag{8.4}$$

Constrained quaternions simplify our boundary curve expressions, thereby improving the performance of the overall technique. Notice that the rotation matrix is a second order expression in two variables. An Euler angle (yaw,pitch,roll) parameterization, on the

other hand, would have produced a fourth order expression in two variables.

Similar to the methods discussed in Chapters 3 and 6, we will be intersecting non-linear curves via resultant techniques [Man92] Resultant elimination techniques are exact in theory, but the accuracy and the running time of these routines depend on the number of free variables and the total degree of the expressions. Essentially, the most compact representation produces the best results, and this is why we used constrained quaternions to characterize out-of-plane rotations. Since the rotation transformation corresponding to constrained quaternions has total degree two, our discretization and correspondence boundary curves have degree at most four; therefore, the dixon resultant polynomials have maximum degree eightteen.

8.4.2 Dixon Resultant

Given two quartic multivariate polynomial expressions $f(a,b), g(a,b)$, we can construct a dixon resultant in the monomial coefficient expressions $bf_i(a), bg_j(a)$.

$$f(a,b) = f_4(a)b^4 + f_3(a)b^3 + f_2(a)b^2 + f_1(a)b + f_0(a) \tag{8.5}$$

$$g(a,b) = g_4(a)b^4 + g_3(a)b^3 + g_2(a)b^2 + g_1(a)b + g_0(a) \tag{8.6}$$

$$Dixon(f,g) = \begin{bmatrix} f_1g_0 - f_0g_1 & f_2g_0 - f_0g_2 & f_3g_0 - f_0g_3 & f_4g_0 - f_0g_4 \\ f_2g_0 - f_0g_2 & f_3g_0 - f_0g_3 + f_2g_1 - f_1g_2 & f_4g_0 - f_0g_4 + f_3g_1 - f_1g_3 & f_4g_1 - f_1g_4 \\ f_3g_0 - f_0g_3 & f_4g_0 - f_0g_4 + f_3g_1 - f_1g_3 & f_4g_1 - f_1g_4 + f_3g_2 - f_2g_3 & f_4g_2 - f_2g_4 \\ f_4g_0 - f_0g_4 & f_4g_1 - f_1g_4 & f_4g_2 - f_2g_4 & f_4g_3 - f_3g_4 \end{bmatrix}$$

8.4.3 Example

In this section, we step through an example in order to more clearly explain the process of computing the intersections of discretization boundary curves. The first curve is an orientation discretization boundary curve where the projected angle between edges e and e' is 3.5 radians: $e = (0,1,1)$ and $e' = (1, 1/4, 1/2)$ and $\theta = 3.5$. The second curve is a length discretization boundary curve where the projected length between two points is 0.5 where $p = (0,0,0)$ and $p' = (3/4, \Leftrightarrow 1/2, \Leftrightarrow 1/4)$.

$$\Pi(E) = (b + a\,b, \frac{1}{2} \Leftrightarrow a \Leftrightarrow \frac{a^2}{2} + \frac{b^2}{2}, 0, \frac{1 + a^2 + b^2}{2})$$

$$\Pi(E') = (\frac{1}{2} + \frac{a^2}{2} + \frac{b}{2} + \frac{a\,b}{4} \Leftrightarrow \frac{b^2}{2}, \frac{\Leftrightarrow a}{2} + a\,b + \frac{1}{8} \Leftrightarrow \frac{a^2}{8} + \frac{b^2}{8}, 0, \frac{1 + a^2 + b^2}{2})$$

$$(\frac{1}{\tan(\theta)}\Pi(E)\cdot\Pi(E')) \Leftrightarrow (\Pi(E)\times\Pi(E')) \quad =$$
$$0.729904 \Leftrightarrow 1.70981\,a + 0.375\,a^2 \Leftrightarrow 0.959808\,a^3 \Leftrightarrow 0.604904\,a^4 + 0.833702\,b + a\,b \Leftrightarrow 0.166298\,a^2\,b$$
$$+0.625\,b^2 \Leftrightarrow 0.959808\,a\,b^2 \Leftrightarrow 1.20981\,a^2\,b^2 \Leftrightarrow 0.166298\,b^3 \Leftrightarrow 0.604904\,b^4$$

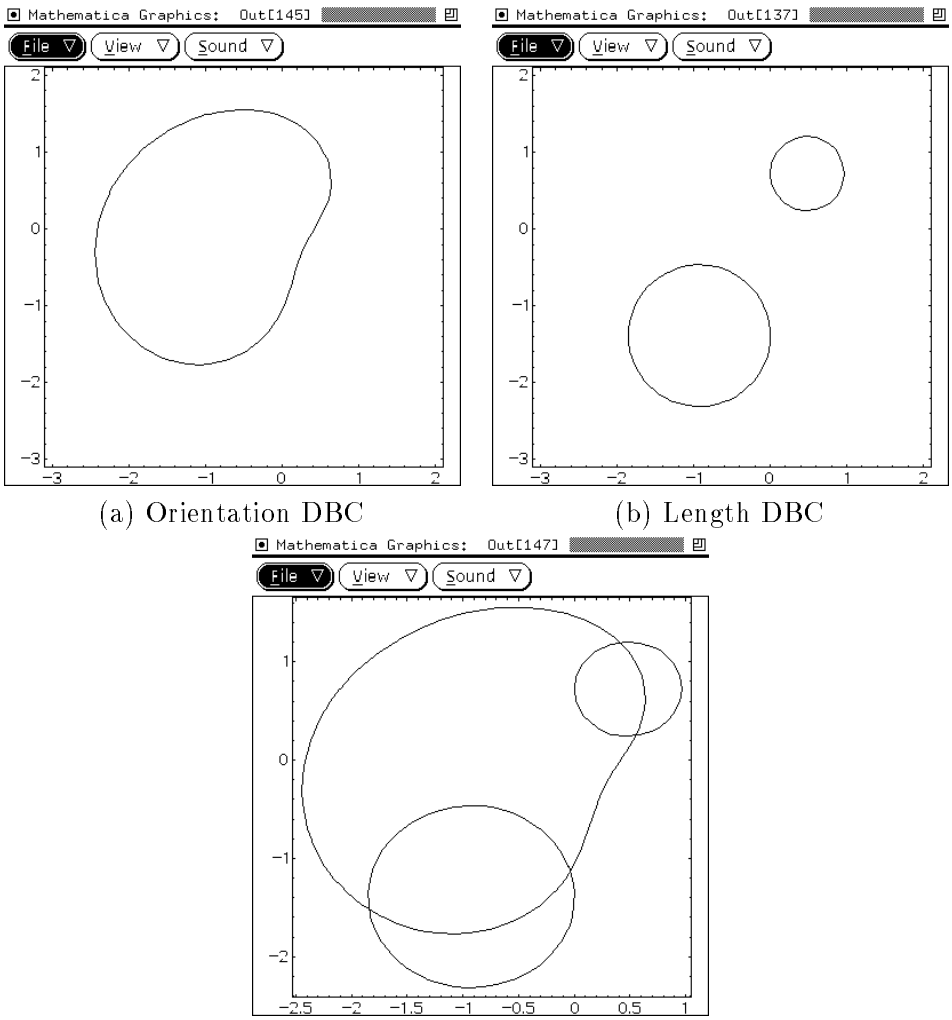
$$\begin{aligned}\Pi(P) &= (0,0,0)\\ \Pi(P') &= (\frac{\Leftrightarrow b}{4} \Leftrightarrow \frac{a\,b}{2} + \frac{3}{8} + \frac{3a^2}{8} \Leftrightarrow \frac{3b^2}{8}, \frac{a}{4} + \frac{3\,a\,b}{4} \Leftrightarrow \frac{1}{4} \Leftrightarrow \frac{a^2}{4} + \frac{b^2}{4}, 0, \frac{1+a^2+b^2}{2})\end{aligned}$$

$$\Pi(E)\cdot\Pi(E) \Leftrightarrow \delta^2(1+a^2+b^2)^2 \quad =$$
$$0.140625 \Leftrightarrow 0.125\,a + 0.09375\,a^2 + 0.125\,a^3 + 0.140625\,a^4 \Leftrightarrow 0.1875\,b \Leftrightarrow 0.75\,a\,b$$
$$+0.1875\,a^2\,b \Leftrightarrow 0.21875\,b^2 + 0.125\,a\,b^2 + 0.28125\,a^2\,b^2 + 0.1875\,b^3 + 0.140625\,b^4$$

The two discretization boundary curves are depicted in Figure 8.15, and we can visually count four intersections.

Now, we step through the process of utilizing resultants to compute the simultaneous solutions to a set of expressions by solving for one variable at a time. The first step involves separating out the monomial coefficients from each of the expressions. Next, we construct the four by four dixon resultant using these monomial expressions, and take the determinant (equation (8.7). This polynomial characterizes all of the roots of a at all of the simultaneous roots of both expressions. Next, we numerically compute the roots of this polynomial $\{a_0, a_1, \ldots\}$ by computing the eigenvalues of a companion matrix. Finally, we backsubstitute all of the real roots $\{a_0, a_1, \ldots\}$ into one of the equations to solve for the other variable $\{b_{a_i,0}, b_{a_i,1}, \ldots\}$ at each of these roots. Finally, we test each candidate intersection $\{(a_i, b_{a_i,j})\}$ using both expressions. The actual intersections are underlined in Table 8.1.

$$\begin{aligned}f_4(a) &= \Leftrightarrow 0.604904\\ f_3(a) &= \Leftrightarrow 0.166298\\ f_2(a) &= \Leftrightarrow 1.20981\,a^2 \Leftrightarrow 0.959808\,a + 0.625\\ f_1(a) &= \Leftrightarrow 0.166298\,a^2 + a + 0.833702\\ f_0(a) &= \Leftrightarrow 0.604904\,a^4 \Leftrightarrow 0.959808\,a^3 + 0.375\,a^2 \Leftrightarrow 1.70981\,a + 0.729904\\ g_4(a) &= 0.140625\\ g_3(a) &= 0.1875\end{aligned}$$



(a) Orientation DBC (b) Length DBC

Figure 8.15: The curves in configuration space representing the discretization boundary curves and the intersections shown by the superimposition of the curves

a	b
-1.8405	{ -1.51896, 1.01774, 0.113154 ± -2.11373 i }
-0.0377304	{ -1.12603, 1.47608, -0.312479 ± -0.832482 }
0.405509	{ 1.19538, -0.0147515, -0.72777 ± 1.07029 i}
0.547508	{ 0.244947, 0.995846, -0.757854 ± 1.2237 i}

Table 8.1: Intersections computed using the dixon resultant

$$
\begin{aligned}
g_2(a) &= 0.28125\,a^2 + 0.125\,a \Leftrightarrow 0.21875 \\
g_1(a) &= 0.1875\,a^2 \Leftrightarrow 0.75\,a \Leftrightarrow 0.1875 \\
g_0(a) &= 0.140625\,a^4 + 0.125\,a^3 + 0.09375\,a^2 \Leftrightarrow 0.125\,a + 0.140625
\end{aligned}
$$

$$
\begin{aligned}
|Dixon(f,g)| &= 0.0357382a^8 + 0.0149104a^7 + 0.027256a^6 + 0.104523a^5 \Leftrightarrow \\
&\quad 0.121598a^4 + +0.122278a^3 \Leftrightarrow 0.0885331a^2 + 0.0185907a + 0.000833594
\end{aligned} \tag{8.7}
$$

$$
\begin{aligned}
Solns = \{ \quad & \Leftrightarrow 1.8405, \Leftrightarrow 0.0377034, 0.405509, 0.547508, \\
& \Leftrightarrow 21052.7 \pm 21052.7i, 0.109221 \pm 1.35547i, 0.144766 \pm 0.893163i\}
\end{aligned}
$$

8.5 Experiments and Results

We constructed complete indexing tables for various polygonal models at various discretizations. Table 8.2 presents the sizes of complete lookup tables generated for various objects using this technique.

Object	Disc. (mm)	n	Table Size	κ	$\sigma(\kappa)$
Cube	0.5	24	20632	1.900	1.287
Cube	0.25	24	84368	1.681	0.868
Box	1.0	24	11233	3.607	3.006

Table 8.2: Sizes, average number of correspondences, standard deviations of the number of correspondences for complete indexing tables for various objects and resolutions.

8.6 Conclusion

In this chapter, we described an application of a complete indexing table construction technique to a machine vision problem. Our indexing system recognized three-dimensional polyhedral objects from pairs of image rays. We presented results concerning the sizes of complete indexing tables for various objects and discretization resolutions.

8.6.1 Extensions to More Complex Projections

The method of identifying objects from pairs of rays is extendible to scaled orthographic projection; this extension involves only extracting the four orientation measurements $(\gamma, \alpha, \beta, \omega)$ and ignoring the distance measurement δ. We can further also extend this approach by incorporating the lengths of some of the ray edges. With the introduction of one additional length measurement, we can distill five indexing coordinates from scaled orthographic projective image data. By incorporating all of the length measurements, we can distill nine indexing coordinates for unscaled orthographic projection, or eight indexing coordinates for scaled orthographic projection.

Chapter 9

Efficient Indexing Techniques for Model Based Sensing

Abstract

Indexing is a model-based recognition technique, in which unknown objects are identified using lookup tables. Indexing coordinates are extracted from sensed features, and the indexing coordinates specify a table entry containing the object's identity. Usually, only a small fraction of the possible indexing coordinates correspond to modeled objects, and hash tables are often used to save space. In this chapter, we present a new indexing data structure called a tree grid *which has two advantages over hash tables: (i) The tree grid preserves spatial ordering, so that nearby indexing entries can be retrieved efficiently (ii) The tree grid compacts the storage size of the table by a factor of as much as two orders of magnitude. For an object with k normalized degrees of freedom, the tree grid consists of a k dimensional table, with a tree attached to each cell, where the tree is ordered by an additional coordinate. In other words, k coordinates index an ordering of the interpretations, and 1 coordinate determines the consistent interpretations for objects with k degrees of freedom.*

We also show that for almost all model sets, $2k+1$ indexing coordinates are sufficient to discriminate between two generic models, implying that $2k+1$ indexing coordinates specify a unique interpretation. We have implemented an indexing algorithm for recognizing 3D objects from pairs of image rays using the tree grid technique, and the results are reported.

9.1 Introduction

State-of-the-art machine vision technology can recognize objects from image data; the next step is real-time model-based recognition. Recognition involves interpreting the sensed features as model features. This could require checking an exponential number of hypothetical interpretations, without heuristics for pruning the search. Indexing is a different approach to solving the correspondence problem; indexing involves precomputing all possible interpretations of the object offline and saving them in a table, and then using the table to look up the object's identity.

Definition 9.1 *Hypothesis H* is a matching between a group of model features and a group of sensed features.

Overconstraint, the property that the number of degrees of constraint exceeds the number of degrees of freedom, is crucial to indexing because each observation not only specifies a configuration for each hypothesis, but the observation also rates the quality of the match between the observation and each hypothesis and configuration as well. The set of indexing points predicted by a model characterizes a lower dimensional subspace embedded in the indexing space. One advantage of indexing is using a single lookup to compare an observation with all of the hypotheses. This is achieved by merging the lookup tables for various hypotheses into a composite lookup table. The two advantages of indexing are simplified online computations and performing only a single lookup operation, which enable real-time performance.

In this chapter, we present an efficient indexing technique, the *tree grid*, which trades off online speed for space compression and spatial faithfulness (order preservation). Lookup time increases by a factor of 10, but table size shrinks by a factor of 100, allowing more tables, of finer resolution, to share fast main memory. Spatial faithfulness enables the user to change the resolution of the search online, to determine the most consistent hypothesis without performing an exponential search. We present experimental results of this technique for a three-dimensional object recognition algorithm from pairs of edge-detected rays.

9.1.1 Previous Work

Clemens and Jacobs proved that in order to recognize model feature groups of three dimensional points under orthographic projection, each model feature group must be represented by at least a two-dimensional surface in a single index space [CJ91], requiring an inordinate number of table entries. Jacobs developed an indexing algorithm for the more general class of affine transformations, which separates the two-dimensional surface in indexing space into two one-dimensional surfaces (lines) in two smaller spaces [Jac92]. This separation reduces the size of the tables and also simplifies table construction. The main drawback of this approach is assumes affine transformations, which have eight degrees of freedom; therefore, sensed feature groups consisting of five image points are necessary to achieve overconstraint.

This work also stems from work in the field of coding theory. A set of similar lists can be stored in $O(e)$ space, where e is the total number of edits involved in synthesizing the lists, which can be much less than the sum of the lengths of the lists. Cole [Col86] presented an algorithm which required examining all of the lists simultaneously. Sarnak and Tarjan [ST86] described an incremental algorithm which precluded memoization. Memoization refers to the process of remembering which sublists have already been characterized by trees, so that the trees can be used to characterize the same sublist in many different localitions. The data structures described by Cole and Sarnak and Tarjan preclude memoization because they affix a time stamp to every root node, and use this time stamp to determine which branch of the tree to follow.

9.1.2 Framework

Definition 9.2 *Configuration* $\vec{x}$ describes the observable state. The configuration corresponds to the object's pose with respect to the camera. *Configuration space* (C) describes the set of all configurations ($\vec{x} \in$ C). For example, under unscaled three-dimensional orthographic projection, configuration space can be defined by coordinate axes of x, y, θ, ϕ, ψ; z is not a coordinate axis because it is unobservable. The observation technique may normalize out observable degrees of freedom; for example, the rotation in the image plane is normalized by rotating a group of model points so that the vector from the first point to the second point is horizontal. k refers to the number of normalized degrees of freedom (C $= \mathbb{R}^k$).

Definition 9.3 *Indexing coordinate* y_i refers to a real valued scalar data value distilled from a group of sensed features; $\hat{y}$ refers to an n-tuple, a set of n simultaneously observed indexing coordinates. *Indexing space* ($\mathbb{I}$) describes the set of all possibly observed n-tuples ($\mathbb{I} = \mathbb{R}^n$, $\hat{y} \in \mathbb{I}$). q image points provide $2q$ indexing coordinates.

Definition 9.4 *Hypothesis function* F_H for a given hypothesis H, provides a map from configuration space into indexing space: $\hat{F_H} : \mathrm{C} \rightarrow \mathbb{I}$. The hypothesis set is the range of this mapping, the set of all indexing coordinates consistent with a particular hypothesis H.

The purpose of indexing is to map *from* the indexing coordinates $\hat{y}$ *to* the configuration $\vec{x}$. Indexing tables $T_H(\vec{y})$ list the configurations $\{\vec{x}\}$ consistent with an n-tuple of indexing coordinates (equation (9.1)).

$$T_H(\hat{y}) = \{\vec{x} | \hat{F_H}(\vec{x}) = \hat{y}\} \tag{9.1}$$

Since each table T_H only covers a small fraction of the indexing space $\mathbb{I}$, many such tables can be merged into a composite table ($T_{\bigcup_{H_i}}$ in equation (9.2)); thereby, one can usually determine both the hypothesis and the configuration consistent with given indexing coordinates in a single lookup operation. Each n-tuple of indexing coordinates $\hat{y}$ usually implies a unique hypothesis $H_{\hat{y}}$ and configuration $x_{\hat{y}}$. Essentially, indexing techniques are used to solve the problems of the form: given an indexing coordinates $\hat{y}$, enumerate all consistent hypotheses H_i and configurations $\{\vec{x_i}\}$ which satisfy $\hat{y} = \hat{F_{H_i}}(\vec{x_i})$ (equation (9.2)).

$$T_{\bigcup_{H_i}}(\hat{y}) = \{< H, \vec{x} > | \vec{F_H}(\vec{x}) = \hat{y}\} \tag{9.2}$$

Indexing techniques are based on the assumption that the predicted indexing coordinates and the sensed indexing coordinates will vary so slightly that both index the same table entry. When each model characterizes a lower dimensional subspace embedded in the indexing space, indexing tables are usually implemented as hash tables. In order to index table entries, the indexing coordinates must first be quantized into integral indices. For completeness and correctness, indexing tables must account for every valid hypothesis and predicted indexing coordinates.

The multiple hypotheses scenario is not uncommon: one example is the task of recognizing modeled objects in an image. The first step involves identifying salient features, such as edge-based features, from the sensor data. The majority of the computation is spent in the second step: interpreting the sensed features groups as groups of modeled features.

Relative to a group of sensed features, each group of model features constitutes a separate hypothesis.

For a system with k normalized degrees of freedom, k data values only constrain the configuration, $k+1$ data values overconstrain the configuration and thereby substantially narrow the set of consistent hypotheses. Furthermore, we rigorously show that $2k+1$ data values are necessary and sufficient in order to specify a unique interpretation for generic models.

Current indexing implementations have three drawbacks: the completeness requirement, the size of the composite tables, and the spatially unfaithful nature of hash tables. Constructing complete tables is a nontrivial task which was discussed in Chapters 2,3, and 8. Indexing table sizes are an important issue because a large composite table may cause the disk to thrash, obviating the theoretical fast performance. Of lesser importance is spatial faithfulness, which simplifies the task of determining the *closest* hypothesis; this is normally accomplished by searching all of the nearby indexing coordinates, a procedure which, for hashed indexing tables, takes time exponential in n, the number of indexing coordinates.

9.1.3 Algorithmic Overview

In this chapter, we present a table indexing technique, the tree grid, which provides compaction and spatial faithfulness. The tree grid achieves these results by ordering the hypotheses, and performing binary searches on these orderings. k indexing coordinates are used to index an ordering of the hypotheses, and 1 indexing coordinate is used to search through that ordering. Storage space is lessened because the hypothesis orderings are similar lists, and the storage space for similar lists depends upon the number of edits, not the total number of elements. This approach exploits the coherence of the hypothesis orderings; we want to reduce the task of indexing to performing a binary search on the hypothesis orderings. In this chapter, we present an ordering on the hypotheses.

The tree grid technique relies on three major ideas: concentration, segmentation, and discretization. We concentrate on $k+1$ data values of the indexing n-tuple. These $k+1$ data values are segmented into k independent coordinates and 1 dependent coordinate. The k independent coordinates are discretized to a grid point, indexing an ordering of the hypotheses. Finally, we determine the hypotheses consistent with the indexing coordinates

by performing a binary search using the dependent coordinate, and then validating each of those hypotheses using the $n \Leftrightarrow (k+1)$ indexing coordinates.

9.1.4 Overview

In section two, we rigorously prove that $2k+1$ data values are necessary and sufficient to specify a unique hypothesis for every configuration of generic models. In section three, we develop a theoretical framework for the tree grid indexing technique. In section four, we detail the implementation of the tree grid indexing technique. In section five, we present experimental results of the storage space compaction of the tree grid technique. Finally, we conclude by highlighting the contributions of this work.

9.2 Sparse Observation Theorem

In this section, we present a rigorous proof of the sparse observation theorem, and we also present two examples of systems which satisfy the theorem's preconditions. This theorem relies upon results in manifold and stratification set theory [GG73].

Theorem 9.1 *$2k+1$ observed values are necessary and sufficient to distinguish between two objects each with k normalized degrees of freedom for generic models in any configuration and the observed values are functionally independent,* i.e., *by perturbing both pose and model parameters, we can move a vector of sensor outputs $\hat{y} \in \mathbb{I}$ a small distance in any direction.*

Proof The outline of the proof is as follows: First we show that if we can move a sensor value $\hat{y} \in \mathbb{I}$ a small distance in any direction, then the manifolds are regular, then, almost every pair of hypotheses intersect transversally. Secondly, we relate that codimension of a transversal intersection of two manifolds or stratified sets H_1 and H_2 is equal to the sum of the codimensions of those stratifications [GG73]. Thirdly, we show by a dimension counting argument that for $n > 2k$, the codimension of intersection $> 2k$, implying that the dimension of the intersection is < 0; *i.e.,* the intersection is an empty set.

Let P be a parametrization of the space of models. Let $M : P \times C \rightarrow \mathbb{I}$ be the *meta-hypothesis function.* So $\hat{y} = M(p, \vec{x})$ is the sensor tuple arising from model p in configuration $\vec{x}$. We can then give an alternative definition for the hypothesis set for a model p as $M(p, C) \subset \mathbb{I}$. Note that we seem to be ignoring the correspondence problem between model and image features. For the purposes of this proof, which relates to dimension, we

are including the feature assignment as part of the model. Since there are finitely many possible correspondences, it does not affect our result whether we do this or not.

Now let $H_1 = M(p_1, C)$ be the hypothesis set for hypothesis p_1, and $H_2 = M(p_2, C)$ be the hypothesis set for hypothesis p_2. Under a certain condition on the model map, and for generic hypotheses, i.e. almost all choices of p_1 and p_2, H_1 and H_2 will be disjoint. If H_1 and H_2 are disjoint, there is no possibility of confusing p_1 and p_2. There are several conditions on M that imply this. The first and strongest is:

Condition 1:
A map $M : P \times C \to \mathbb{I}$ is *regular* if its differential (jacobian) is everywhere surjective. This condition is stronger than necessary, but is easy to state, and probably easier to verify in most situations. In essence it says that by perturbing both pose and model parameters, we can move a sensor value $\hat{y} \in \mathbb{I}$ a small distance in any direction.

Condition 2:
A map $M : P \times C \to \mathbb{I}$ is *transversal* to a stratified set H_1 if M is transversal to all the strata in H_1.

For definitions of transversality, we refer the reader to [GG73]. Condition 1 implies condition 2 for any H_1, so it is strictly stronger, and we take it as the premise for our theorem:

Theorem 9.2 *Suppose condition 2 above holds for a hypothesis function (mapping) M, and that the dimension of the indexing space is at least $2k+1$. Then for almost all choices of $p_2 \in P$, $H_2 = M(p_2, C)$ is disjoint from H_1.*

Proof: A basic property of transversality is that if a parametrized class of mappings (M in this case, parametrized by P) is transversal to a manifold (the manifolds in H_1), then almost every map in the class is transversal to H_1. See e.g. the remark before Corollary 4.7 of [GG73].

Now M is transversal to H_1 by condition 2. Therefore $M(p_2, \cdot) : C \to \mathbb{I}$ is transversal to H_1 for almost every p_2. The transversality condition implies that the codimension of $M(p_2, \cdot)^{-1}(H_1)$ equals the codimension of H_1 in $\mathbb{I}$. Since H_1 has dimension k and O has dimension $2k+1$, the codimension of H_1 is $k+1$. The codimension of $M(p_2, \cdot)^{-1}(H_1)$ is also $k+1$, but it inhabits configuration space C of dimension k, which means it must be empty. $M(p_2, \cdot)^{-1}(H_1)$ empty says that there is no point in both H_1 and H_2. $\square$

9.2.1 Examples

The sparse observation theorem is widely applicable to many systems. Towards this end, we present two systems for which we can always move a sensor value $\hat{y} \in \mathbb{I}$ a small distance in any direction by perturbing model and pose parameters. In order to show the limits of this theorem, we also present a system for which the precondition does not hold.

Figure 9.1 shows a system where five indexing coordinates are extracted from a pair of rays under unscaled orthographic projection. We can normalize away the rotation and translation in the image plane by insisting that v_1 lie at the origin and that one of its edges lie along the x axis. This leaves only two degrees of freedom: the out of plane rotations; $C \subseteq S^2$. The indexing coordinates $\mathbb{I} \subseteq R^5$ are the x, y position of v_2, and α, β, γ, the orientations of the other rays relative to the x-axis. Notice that we can always vary each of the five indexing coorindates independently by perturbing the model. We can modify x, y by modifying v_2's position, and we can modify α, β, and γ by independently modifying the orientations of the edges.

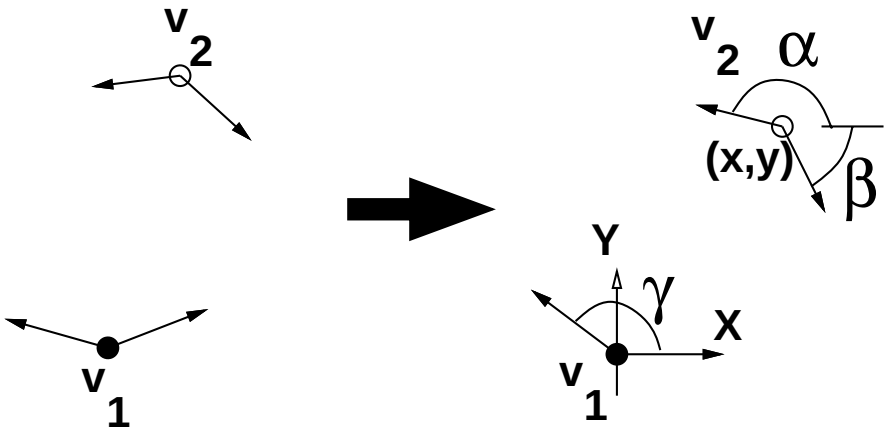


Figure 9.1: For two rays v_1 and v_2, after rotating and translating vertex v_1 to the origin such that one of its rays is aligned with the x-axis, the indexing coordinates are the x, y position of v_2, and α, β, γ, the orientations of the other rays relative to the x-axis.

The crossbeam sensing technique discussed in Chapter 2 also satisfies the preconditions of the sparse observation theorem. Figure 9.2 describes a system where $2b \Leftrightarrow 2$ parallelepiped parameters are extracted from a planar object by b line probes. These line probes provide b diameters, the distance between opposing parallelpiped edges, and $b \Leftrightarrow 2$ inset distance measurements, which characterize the distance between the intersection of

two extended edges of the parallelepiped measured normal to a third edge. Since the parallelepiped's position is irrelevant, the system only has a single degree of freedom: rotation $C = S^1$. The indexing coordinates $\mathbb{I} \subseteq R^{2b-2}$ are the b diameters and $b \Leftrightarrow 2$ inset distances. Notice that we can always vary each of the indexing coorindates independently by perturbing the model. We can independently modify the diameters by moving the vertices which do not correspond to the inset distance measurements, and we can independently modify the inset distances by moving all of the vertices which do correspond to the inset distance measurements. Even when two parallelepiped edges are defined by the same vertex, we can independently modify the edge positions by moving the vertex along either of the edges.

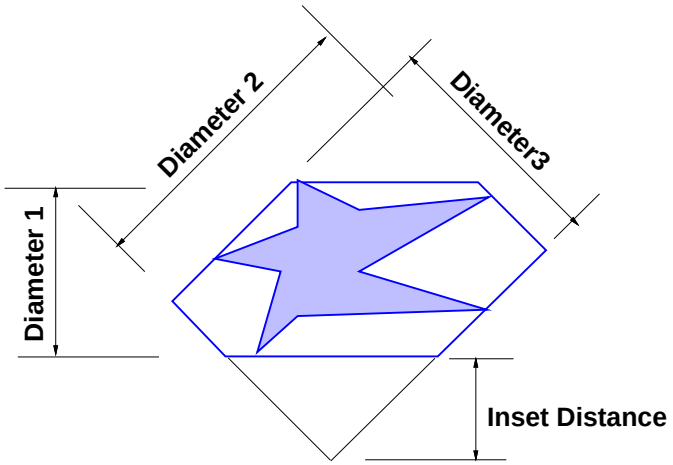


Figure 9.2: The crossbeam sensing system described in Chapter 2 also satisfies the preconditions of the sparse observation theorem. The b diameters and $b \Leftrightarrow 2$ inset distances can be independently varied by suitably shifting the object vertices.

For polygonal models, the scanning sensing technique discussed in Chapter 3 does not satisfy the preconditions of the sparse observation theorem. Figure 9.3 describes a system where $2b$ scanline endpoints are extracted from a planar object by b point probes. Notice that if any three of the scanline endpoints correspond to the same linear model feature, that we can not vary the scanline endpoints independently (since the feature provides a linear constraint on the scanline endpoints). Interestingly, the failure to satisfy the preconditions is due to the model class, and not the underlying system; the system of scanning objects with unconstrained boundaries does satisfy the preconditions of the sparse observation theorem.

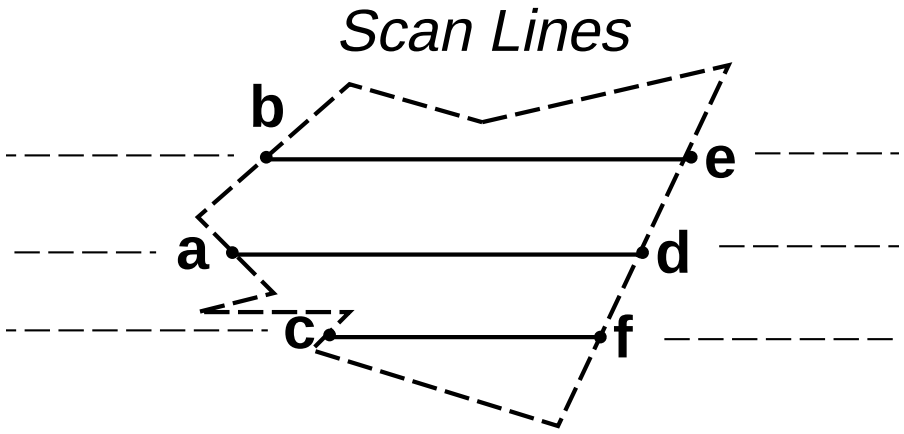


Figure 9.3: The scanning beam sensing system described in Chapter 3 does not satisfy the preconditions of the sparse observation theorem.

9.3 Theoretical Framework

In the tree grid indexing technique, consistent hypotheses are found using only $k+1$ of the n indexing coordinates. The projection $\Pi : \mathbb{R}^n \rightarrow \mathbb{R}^k$ extracts k indexing coordinates, and the projection $\overset{k+1}{\Pi}: \mathbb{R}^n \rightarrow \mathbb{R}^{k+1}$ which extracts $k+1$ coordinates.

Definition 9.5 *Independent coordinates* correspond to k chosen indexing coordinates ($\Pi(\hat{y})$) such that $F(C)$ is surjective onto $\mathbb{I}_{Ind}$ the subspace of $\mathbb{I}$ spanned by the independent coordinates. In this chapter, we assume that the first k indexing coordinates are independent. k independent coordinates provide sufficient constraint so that only a finite number of configurations are consistent with $\Pi(\hat{y})$. In figure 9.4, y_1 and y_2 were chosen as the independent coordinates.

Definition 9.6 *Dependent coordinate* is another indexing coordinate (y_3 in Figure 9.4). The independent indexing coordinate $\mathbb{I}_{Dep}$ combined with the dependent indexing coordinates characterize fiber bundles which span the subspace $\mathbb{I}_{Ind} \times \mathbb{I}_{Dep}$. The independent indexing coordinate is chosen such that there are generically a finite number of intersections between each fiber and the hypotheses sets. Therefore, only a finite number of dependent coordinates are consistent with the independent coordinates.

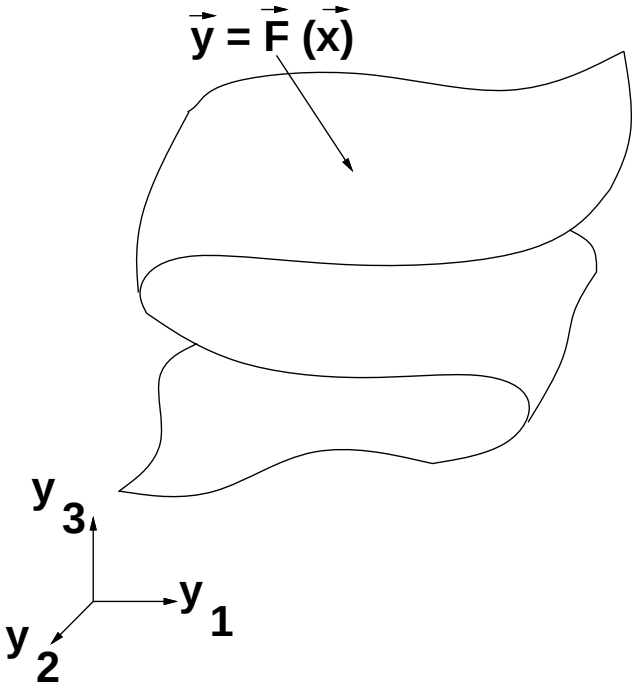


Figure 9.4: A two-dimensional hypothesis set parameterized as $\hat{y} = \hat{F}(\vec{x})$. In this case, I is three-dimensional (y_1, y_2, y_3), and C is two-dimensional (x_1, x_2).

9.3.1 Separating Sets Into Monotone Sheets

In order to realize a unique mapping from independent to dependent coordinates, we want to impose monotonicity. This is achieved by separating the hypothesis sets into monotone hypothesis sheets. On these hypothesis sheets, the dependent coordinate is a single valued function of the independent coordinates.

Definition 9.7 *Hypothesis sheet* $\mathcal{S}_H$ is a continuous subset of the hypothesis set such no two points in the sheet share the same independent coordinates (Figure 9.5).

Base Grid

Indexing into tables requires integral indices; therefore, the indexing coordinates need to be discretized.

Definition 9.8 *Base grid*, $\mathcal{B}$ refers to a k-dimensional grid in $\mathbb{R}^k$. The term *grid point* ($\lfloor \Pi(\hat{y}) \rfloor \in \mathcal{B}$) refers to the discretized independent coordinates $(\lfloor \frac{y_1}{\delta_1} \rfloor, \lfloor \frac{y_2}{\delta_2} \rfloor, \ldots, \lfloor \frac{y_k}{\delta_k} \rfloor)$. The

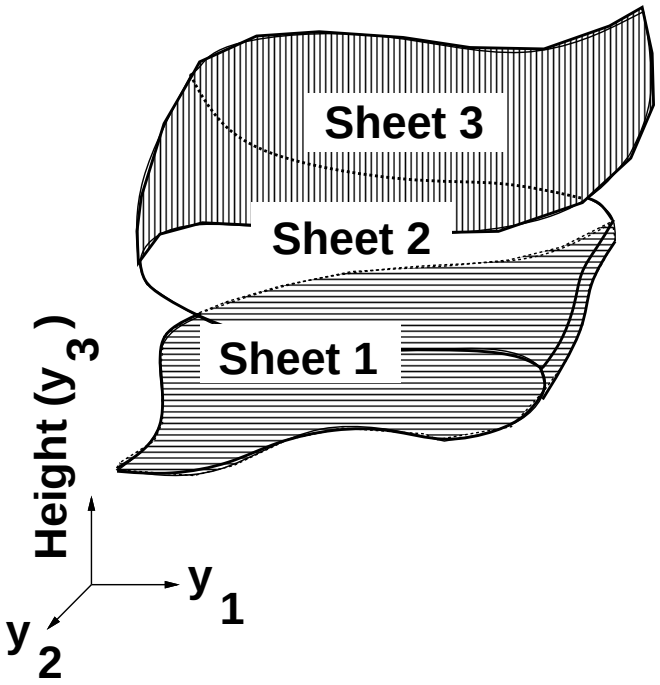


Figure 9.5: The hypothesis set is separated into sheets such that no two points on a sheet share the same independent coordinates.

term M refers to the maximum number of hypothesis sheets over any point on the base grid.

Definition 9.9 *Height function*, $(h(\lfloor \Pi(\hat{y}) \rfloor))$ formulates the dependent coordinate y_{k+1} as a function of the independent coordinates $\lfloor \Pi(\hat{y}) \rfloor$.

Definition 9.10 *Ordered hypothesis list* $(A_{\lfloor \Pi(\hat{y}) \rfloor})$ describes an ordering of the hypothesis sheets at a particular grid point $\lfloor \Pi(\hat{y}) \rfloor$ according to their dependent coordinates (refer Figures 9.6 and 9.7).

9.4 Indexing Table Implementation

In this section, we describe a table indexing implementation technique, the tree grid, which reduces the storage size of the table and preserves order (spatial faithfulness). In the first steps, only $\overset{k+1}{\Pi}(\hat{y})$ is used. For each hypothesis sheet $< H_i, j >$ (j an sheet index of hypothesis H_i), $\Pi(\hat{y})$, the independent coordinates specify a unique configuration (refer

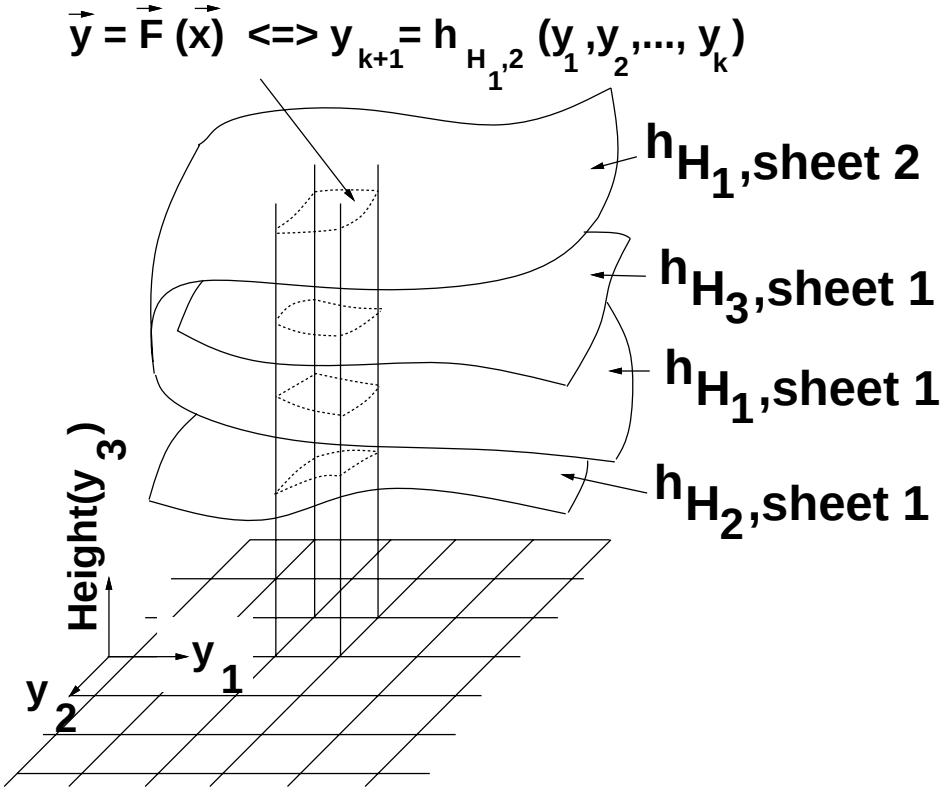


Figure 9.6: The hypothesis sheets above a particular base grid point $\lfloor \Pi(\hat{y}) \rfloor$ are ordered according to their dependent coordinates.

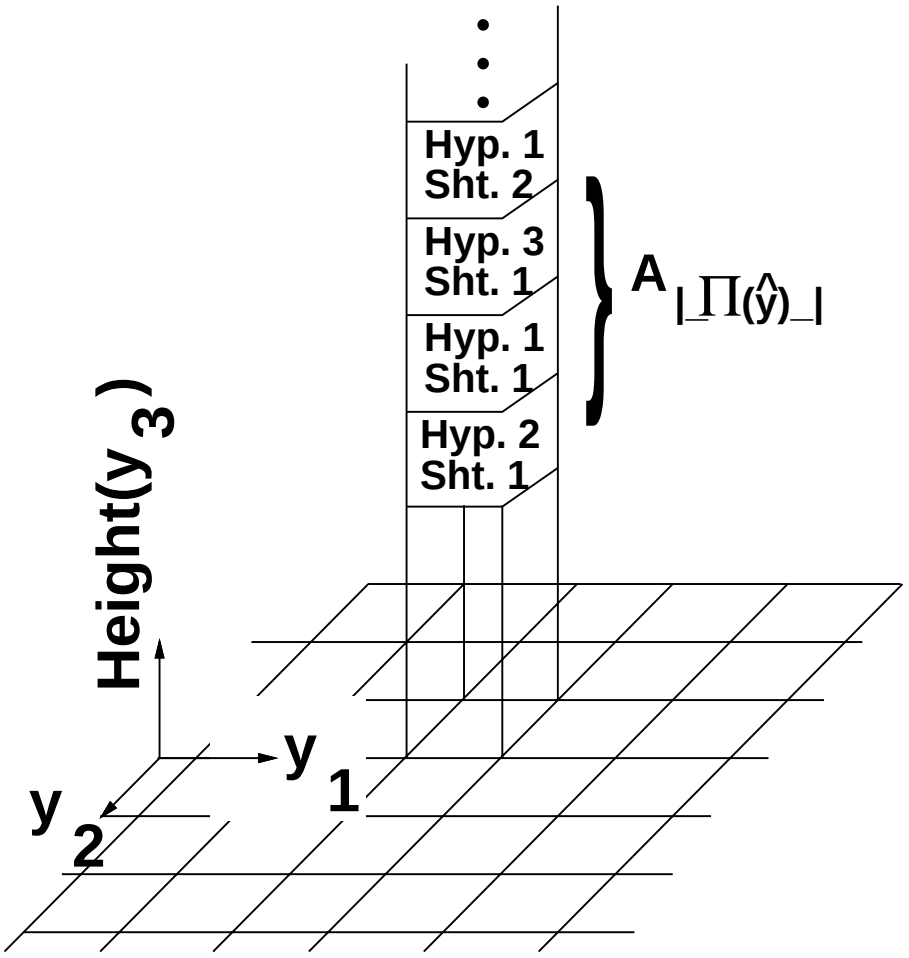


Figure 9.7: The ordered hypothesis list $A_{\lfloor \Pi(\hat{y}) \rfloor}$ of the hypothesis sheets above grid point $\lfloor \Pi(\hat{y}) \rfloor$.

equation (9.3), implying that there exists a function $\hat{F_H}^{-1} : \Pi(\hat{y}) \rightarrow \vec{x}$ (refer equation 9.4). Therefore, there also exists a height function $h_{H_i,j}$ which is the composition of $\hat{F_H}^{-1}$ and f_{k+1}, so that the 1 dependent coordinate y_{k+1} validates the hypothesis sheet and that configuration. The main idea is that we define ordered lists $(A_{\lfloor \Pi(\hat{y}) \rfloor})$ of the hypothesis sheets above all of the grid points $\mathcal{B}$. At runtime, the consistent hypotheses are determined by performing binary search on $A_{\lfloor \Pi(\hat{y}) \rfloor}$ with y_{k+1}.

$$y_1, y_2, \ldots, y_k \rightarrow (x_1, x_2, \ldots x_k)_j \tag{9.3}$$

$$\exists j \; s.t. \; y_{k+1} = f_{k+1}((x_1, x_2, \ldots x_k)_j) \tag{9.4}$$

$$y_{k+1} = h_{H_i,j}(y_1, y_2, \ldots, y_k) \tag{9.5}$$

9.4.1 Sketch of Algorithm

Given a n-tuple of indexing coordinates $\hat{y}$, we discretize k values ($\lfloor \Pi(\hat{y}) \rfloor$) to index an ordered hypothesis list ($A_{\lfloor \Pi(\hat{y}) \rfloor}$), and then search for y_{k+1} in that ordered list. This produces a subset of hypotheses consistent with $\overset{k+1}{\Pi}(\hat{y})$ which are then validated with respect to the entire observation $\hat{y}$.

9.4.2 Ordered Hypotheses Lists

This algorithm is based upon the ordered hypothesis lists $A_{\lfloor \Pi(\hat{y}) \rfloor}$ in order to exploit the coherence (slowly varying nature) of those hypothesis orderings. This strategy saves space because storing similar lists takes up less space than storing the lists individually (equation (9.6)). This strategy also provides spatial faithfulness.

$$\sum_{b \in \mathcal{B}} \min_{\nu \in neighbors(b)} |A_b \Leftrightarrow A_\nu| \ll \sum_{b \in \mathcal{B}} |A_b| \tag{9.6}$$

The only drawback in the tree grid approach is that it involves performing a binary search on the hypothesis sheets at runtime; the search time can be bounded ($O(\log(M))$) by using balanced trees (see Figure 9.4.2). Executing the search involves recomputing the heights $h_{H_i,j}(\lfloor \Pi(\hat{y}) \rfloor)$ of $\log(M)$ hypothesis sheets.

9.4.3 Constructing Space Efficient Ordering Trees

Definition 9.11 *Ordering trees* ($\mathcal{T}_p$) describe the ordering A_p above a grid point p.

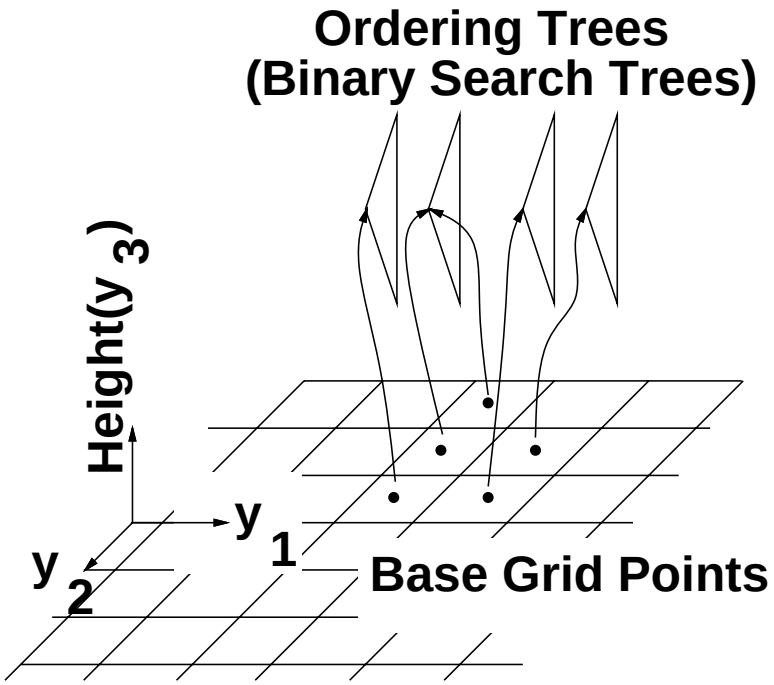


Figure 9.8: Height ordering trees above the base grid

Ordering trees are generated in the following manner in order to share subtrees: $\mathcal{T}_{p_{child}}$, for hypothesis ordering $A_{p_{child}}$, is generated by performing a sequence of tree operations (change, insertion, deletion) on a *parent* tree ($\mathcal{T}_{p_{parent}}$) with a similar hypothesis ordering $A_{p_{parent}}$; each edit corresponds to one tree operation. Notice that each tree operation only affects one path through the ordering tree, allowing both trees to share the other subtrees (see Figure 9.9).

Ordering trees were implemented using persistent tree structures so that tree operations on $\mathcal{T}_{p_{parent}}$ left $\mathcal{T}_{p_{parent}}$ intact. Persistent AVL trees were used rather than a more specialized data structure [ST86] even though each AVL edit generates $O(\log(M))$ new nodes (compared to $O(1)$ new nodes/edit). The total number of nodes to store all of the ordered lists is approximately $O(e_\Sigma \log(M))$, where e_Σ is the total number of edits between the hypothesis orderings.

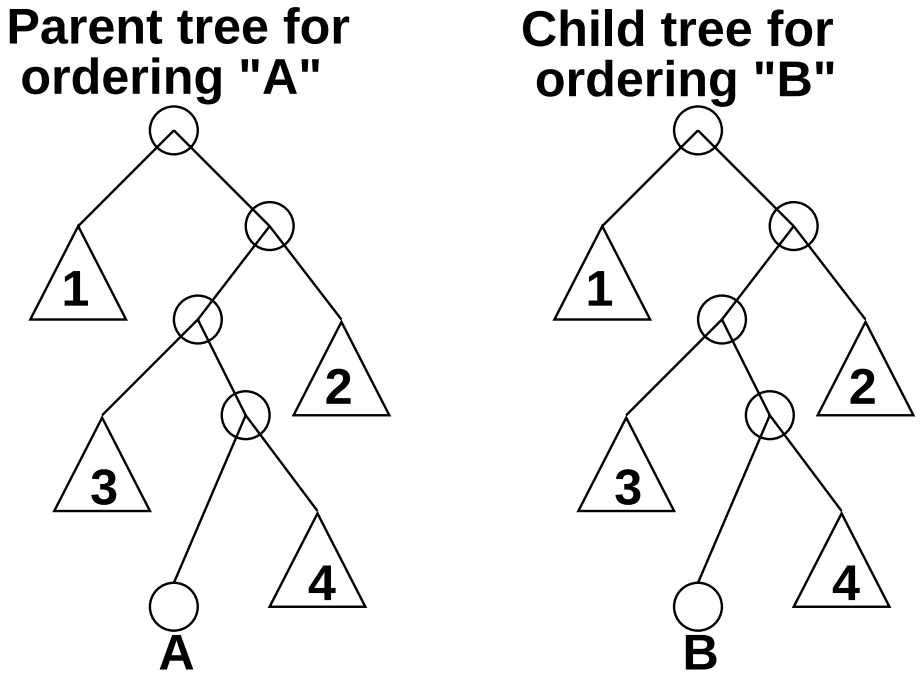


Figure 9.9: An ordering tree $\mathcal{T}_B$, for ordering B, shares many subtrees with $\mathcal{T}_A$, for ordering A, because A and B differ by only a single element. Describing $\mathcal{T}_B$ using $\mathcal{T}_A$'s subtrees involves one edit operation, requiring $O(\log(M))$ nodes.

9.4.4 Indexing Algorithm

The algorithm determines the hypothesis sheet closest to a projected indexing coordinate $\overset{k+1}{\Pi}(\hat{y})$ in $O(\log(M))$ time.

1. Given $\hat{y}$, lookup the ordering tree $\mathcal{T}_{\lfloor \Pi(\hat{y}) \rfloor}$.

2. Determine the hypothesis sheet(s) consistent with the projected indexing coordinates $\overset{k+1}{\Pi}(\hat{y})$ by performing a binary search on the ordering tree $\mathcal{T}_{\lfloor \Pi(\hat{y}) \rfloor}$ using the dependent coordinate, y_{k+1}.

3. Validate each hypothesis using $\hat{y}$.

9.4.5 Generating the Ordering Trees $\mathcal{T}$

1. Compute the minimum edit distances between hypothesis orderings $A_b, A_{b'}$ of neighboring grid points $b, b' \in \mathcal{B}$.

2. Construct a graph G in which grid points are nodes and the edges between neighboring grid points are weighted by the minimum edit distances.

3. Compute the minimum spanning tree M_G of G.

4. Select an initial grid point p_{init} and construct an ordering tree $\mathcal{T}_{p_{init}}$ for hypothesis ordering $A_{p_{init}}$.

5. Perform depth first search on M_G starting with p_{init}. When expanding node p_{parent}, for all previously unexplored neighbors p_{child} construct the ordering tree $\mathcal{T}_{p_{child}}$ by performing a series of edit operations on $\mathcal{T}_{p_{parent}}$ determined by the minimum edit sequence between $A_{p_{child}}$ and $A_{p_{parent}}$.

9.4.6 Analysis

Tree Grid Construction Running Time

The majority of time constructing the trees is spent in step 2, computing all of the minimum edit distances between hypothesis orderings of neighboring grid points, and step 5, modifying trees using the minimum edit sequences between hypothesis orderings of neighboring grid points. Computing the minimum edit distance between two lists of length

M takes Me time, where e is the number of edits. Computing all of the minimum edit distances for $k|\mathcal{B}|$ pairs of hypothesis orderings (we need to check k neighbors for each grid point) takes $k|\mathcal{B}|\times M\overline{e}$ time, where $\overline{e}$ refers to the average number of edits between hypothesis orderings ($|\mathcal{B}|\overline{e} = e_\Sigma$). Recomputing the minimum edit sequences for tree operations takes $|\mathcal{B}| \times M\overline{e}$ time. The algorithm's total time complexity is $O(|\mathcal{B}|kM\overline{e})$. The total number of nodes to describe the tree grid is $e_\Sigma \log(M)$ using AVL trees, or e_Σ using more specialized data structures. e_Σ is bounded by the number of different hypothesis orderings.

Bounds on the Number of Orderings

Next we derive an upper bound on the number of different height orderings for M hypothesis sheets above the grid points. We invoke a theorem which says that M algebraic surfaces of constant degree in $\mathbb{R}^k$ produce an arrangement with $O(M^k)$ cells. This analysis depends upon the assumption that the intersection of each pair of hypothesis-sheets is a surface of constant degree in $\mathbb{R}^k$.

Orderings change in only three ways: removing a hypothesis sheet from the current ordering, inserting a hypothesis sheet into the current ordering, and interchanging the order of two hypothesis sheets. We assume that the majority of orderings result from interchanging the order of two hypothesis sheets.

Since we are only concerned with orderings above the base grid points, we can restrict our view to base lines including rows of grid points ($y_1 = r_1\delta_1, y_2 = r_2\delta_2\ , \ldots |\ r_1, r_2, \ldots \in \mathbb{I}$) Consider the height functions of the models above each of these base lines as one-dimensional curves in the two-dimensional cross-section. Areas of constant ordering correspond to cells in the arrangement of these M height function cross-section curves. The theorem says that the number of cells in the arrangements, and thereby the number of edits, for M such curves is $O(|\mathcal{B}|^{\frac{k-1}{k}}M^2)$.

9.4.7 Efficient Performance Heuristics

Separating Hypotheses Sheets

Dividing the hypotheses sheets into l subsets shrinks the composite table because there are fewer edits, and each edit involves fewer nodes. If a single set contains all of the hypothesis sheets, then there are $O(M^2)$ possible different hypothesis orderings, but if the hypothesis sheets are separated into l subsets, then there are only $O(l(\frac{M}{l})^2)$ different

hypothesis orderings. For a single set, each AVL edit involves $O(\log(M))$ nodes; for l subsets each AVL edit involves $O(\log(M) \Leftrightarrow \log(l))$ nodes. Dividing the hypotheses sheets into subsets increases the lookup time by a factor of l, since each subset must be individually checked.

Memoization

A naive implementation of tree operations generates subtrees for every new subsequence, *even if these subsequences already have been transcribed into trees.* The motivation for memoizing the subsequences is that memoization can only shrink the table size, and memoization takes very little time and space. Checking the balanced subtrees of a sequence with n elements takes linear time. Memoizing the hypothesis ordering subsequences results in table compaction by a factor of between three and ten.

9.5 Experiments and Results

In this section, we describe a model-based three-dimensional recognition technique from two-dimensional image data assuming unscaled orthonormal projection. Observations were generated from pairs of rays in the manner shown in Figure 9.10, an extension of the vertex pairs described by Thompson and Mundy [TM87]; image ray pairs are symmetric to groups of four image points. The rays parameterization implicitly normalizes out three of the five degrees of freedom of unscaled orthographic projection: absolute x and y position information is normalized because vertex v_1 is translated to the origin, absolute orientation is normalized because we assume that the first ray of v_1 is transformed to be parallel to the x-axis. In this manner, there are two normalized degrees of freedom ($k = 2$): θ_x, rotation around the x-axis, and θ_y, rotation around the y-axis. The scale factor could be normalized away by scaling v_1 to length 1.

For a given hypothesis (m_1, m_2), the four configurations $\{(\theta_x, \theta_y)_i\}$ can be extracted as follows:

1. Determine the two orientations $\theta_{x_1}, \theta_{x_2}$ which rotate the vertex m_2 to the correct y position $(v_2.y)$. Rotate m_2 by $\theta_{x_1}, \theta_{x_2}$ to predict m_2' and m_2''.

2. For m_2' and m_2'', determine the two orientations $\theta_{y_1}, \theta_{y_2}$ which rotate m_2' or m_2'' to the correct x position $(v_2.x)$.

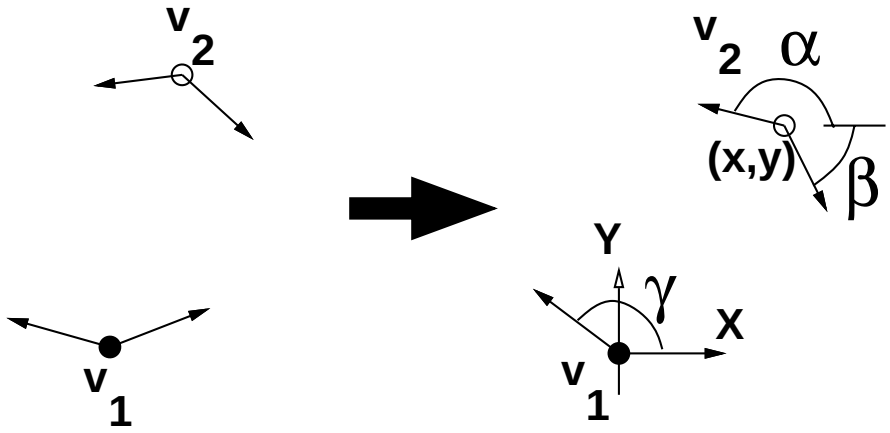


Figure 9.10: After normalizing the rays by rotating and translating vertex v_1 to the origin and one of its rays is aligned with the x-axis, the image ray parameterization is: x, y position of v_2, and α, β, γ, the orientations of the other rays relative to the x-axis.

The dependent coordinate can be any of the orientations of the other rays. Notice that five values $(2k+1)$ are observed, which is the minimum number of indexing coordinates. This should sufficiently differentiate the hypotheses and ensure a unique interpretation. The index table implementation was tested by using real model groups (the asterisk * denotes ray pairs from the stapler in [CJ91]) and random model groups, and then constructing the table for the respective hypothesis sheets. The size of the compacted tables, and the compaction ratio as a function of the number of hypothesis, and resolution (grid spacing) is given in Table 9.1. This recognition technique was experimentally validated by interpreting groups of four image points as groups of stapler model vertices from data given in [CJ91].

9.5.1 Discussion

Notice that increasing the resolution increases the compaction ratio, and increasing the number of hypotheses decreases the compaction ratio. The case of increasing resolution can be explained in the following way: if increasing the resolution does not significantly increase the total number of edits, e_Σ because most of the hypothesis orderings already appear at the coarser resolution, while increasing the resolution significantly increases the number of table entries, the compaction ratio increases. The decreasing compaction for larger hypothesis sets can be explained in the following way: the number of orderings increases as the square of the number of hypotheses; if there are more orderings, each grid

M	$\sqrt{\lvert\frac{1}{\mathcal{B}}\rvert}$ Grid Spacing	# Entries	# Nodes	Compaction
25*	$\frac{1}{100}$	1959460	16983	115.38
25*	$\frac{1}{150}$	4399560	18417	238.89
50	$\frac{1}{100}$	3520520	26808	131.32
50	$\frac{1}{150}$	7902660	28189	345.72
50	$\frac{1}{100}$	2852068	18639	153.02
75	$\frac{1}{100}$	4772700	82156	58.09
75	$\frac{1}{150}$	10631948	105477	100.79
100	$\frac{1}{100}$	6416292	200033	31.28
100	$\frac{1}{100}$	6564076	220021	38.97

Table 9.1: Compaction ratios for lookup tables of real (*) and randomly generated hypotheses. The hypothesis sheets are separated into eight subsets.

point has a higher probability of being associated with a different ordering, requiring more edits. Since the table storage size depends upon the number of edits, the compaction ratio decreases with more hypotheses. Increasing the number of subsets counteracts increasing the number of hypotheses because separating the hypothesis sheets into more subsets improves compaction.

9.6 Conclusion

In this chapter, we presented a indexing table technique, the tree grid, which combined space compaction with spatial faithfulness. This technique exploited the fact that $k+1$ observation values prune the set of consistent hypotheses; we also presented a proof on the minimum number $(2k+1)$ of observation values necessary to specify uniquely a generic hypothesis in any configuration. The tree grid approach enables the user to trade-off storage space and lookup time via a binary search on the hypotheses. In some cases, this approach compacted the total storage size of the table by a hundred-fold. We presented experimental data for three-dimensional object recognition from two image rays.

Chapter 10

Duality of Calibrating Point Scanning and Line Probing Sensors

Abstract

Simple binary sensor systems are capable of handling many industrial tasks currently handled by machine vision systems, and simple sensors have higher performance and are usually easier to set up and calibrate than complex sensors due to their simpler specifications. In this chapter, we present calibration techniques for four different types of sensors: mobile and stationary point scanning and line probing sensors. These calibration methods involve enumerating robot configurations for which a binary sensor detects an object. The sensor's position is determined from the robot configurations using a combination of algebraic and numerical methods. We observed two dualities intrinsic in these computations allowing us to use a single numerical solver routine to compute the sensor position for all four different sensor types. Using these techniques, we calibrated various sensors and then measured the accuracy of these sensor position estimates using self validation techniques. We found sensor calibration accurate to 0.025 *millimeters.*

10.1 Introduction

Object recognition and object localization are fundamental problems in industrial automation. While most research has focused on camera-based approaches because of their many advantages, such as high data density, and commodity pricing and performance characteristics, camera-based methods also have a number of drawbacks, including the difficulty of interpreting image data, and limited pixel precision. For these reasons, there has been a renewed interest in simple sensors and techniques.

Recently, researchers demonstrated that simple sensors and actuators can perform various industrial tasks [WCM93a; WC95b; PC94] (Chapters 2,3); as a consequence of their simple specification, simple sensors can achieve higher performance in terms of speed, accuracy, and robustness than complex sensors, such as video cameras.

In this chapter, we describe a method for planar calibrating sensor and object parameters from accurate planar robot configurations (X_R, Y_R, θ_R) where the sensor detects an object. This approach assumes that the manipulator can provide accurate positional information, although, if the positional discrepancies are normally distributed, high accuracy can be achieved via a large number of data points. For example, to localize a point sensor, a sensor which detects the absence or presence of an object at a particular point, we pass a straight linear object over the sensor and recording when the object was detected; in this chapter, we discuss a methodology from determining the sensor position from such data. This approach involves partial information, because we only know that the sensor detects some point on the object, but we do not know which particular point is detected.

Performance and cost are such paramount concerns that people often overlook secondary concerns, such as ease of set up and calibration. In terms of secondary costs, simple sensors have a distinct advantage over complex sensors due to their relatively simple calibration. Camera based machine vision systems, for example, are complex and inordinately difficult to calibrate.

In this chapter, we detail techniques for calibrating binary beam sensors used for object localization and recognition. We focus on two sensing modalities: line probing and point scanning (refer Figures 10.1,10.2,10.3). Wallack *et al.* utilized a crossbeam sensor (line probing) to recognize cross sections of generalized polyhedral objects in five microseconds and to localize objects to an accuracy of 0.025 mm in a few milliseconds [WCM93a] (Chapter 2). Wallack and Canny utilized a scanning beam sensor (point scanning) to recognize flat

polygonal and polyhedral objects in one hundred microseconds and to localize objects to an accuracy of 0.025 mm in a few milliseconds [WC95b] (Chapter 3).

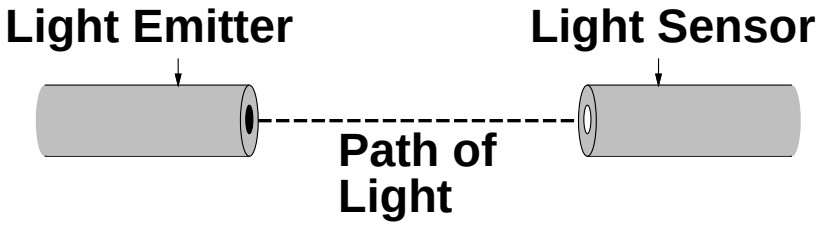


Figure 10.1: A through beam sensor corresponding to a line probing sensor

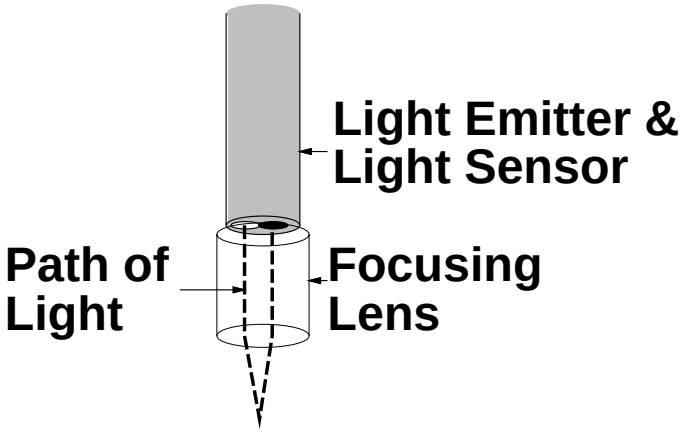


Figure 10.2: A lensed beam sensor corresponding to a point scanning sensor

In order to recognize and localize objects in the workspace as well as objects held in the end effector, we calibrate both *mobile sensors*, sensors attached to the end effector, and *stationary sensors*, sensors fixed in the workspace (refer Figure 10.4).

We assume that the sensor and object parameters are unknown and that the robot configurations (X_R, Y_R, θ_R) are accurate. This approach is similar to the concept of closed loop kinematic calibration by Bennett and Hollerbach [BH91], in which kinematic parameters are computed from joint angles alone. Bennett and Hollerbach dealt with the more difficult problem of three dimensional calibration.

One might assume that calibrating sensors solely from robot positions would require a different numerical routine for each different type of sensor and sensing modality. It turns out that only a single numerical routine suffices. Hill climbing strategies would be susceptible to local extrema since the associated problem is non-linear. Therefore, we use

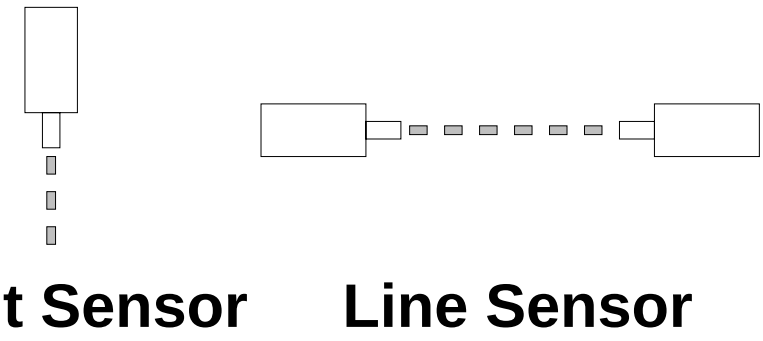


Figure 10.3: Side View: The point sensor is used for scanning object's silhouettes, and the line sensor is used to probe an object's cross section.

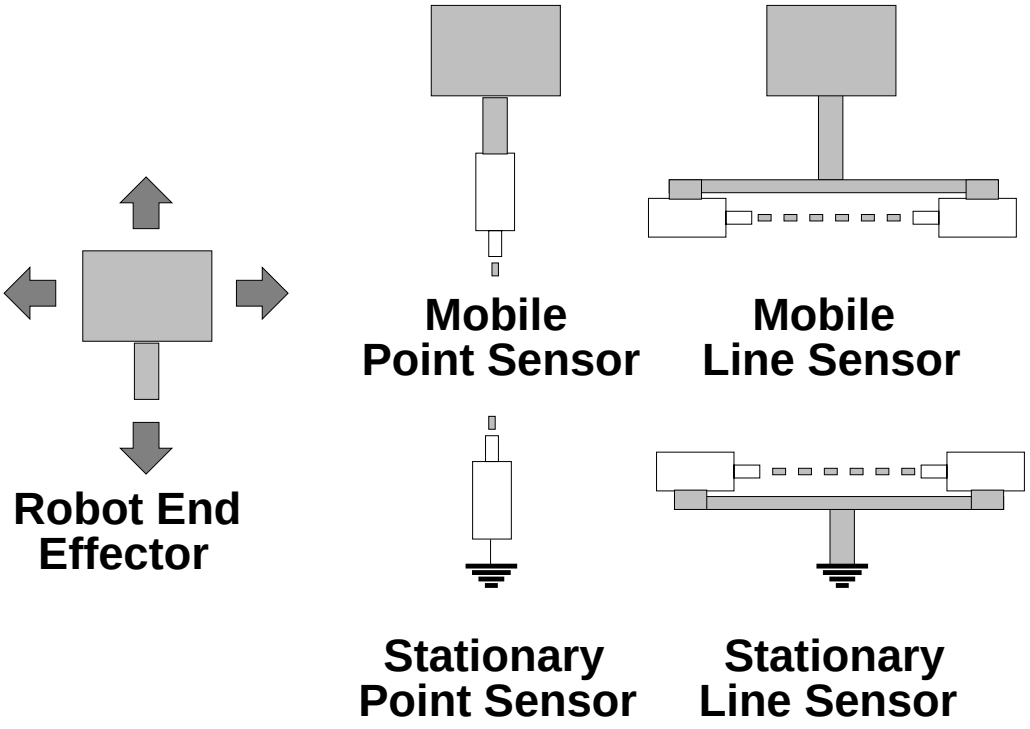


Figure 10.4: Side View: The point and line sensors can be either mobile (attached to the end effector) or stationary (rigidly fixed in the workspace)

a routine based upon a global numerical technique described by Manocha [Man92] (Chapter 4) for solving simultaneous systems of algebraic equations, by reducing the problem to solving a matrix resultant, and then further reducing it to a companion matrix eigenvalue problem.

10.1.1 Calibration Technique

Calibration involves enumerating robot configurations for which an unmodeled point or linear object triggers the sensor. Since both the position of the object and the sensor are unknown, calibration involves not only localizing the sensor, but the object as well. For this reason, we use easily modeled objects: points and lines. Point sensors are calibrated using linear objects (refer Figures 10.5(a) and 10.6(b)) and line sensors are calibrated using point objects (refer Figures 10.5(b) and 10.6(a)). The computational task, therefore, is to determine the position of the line and point solely from robot configurations in which the point (line) object occludes the line (point) sensor (refer Figure 10.7).

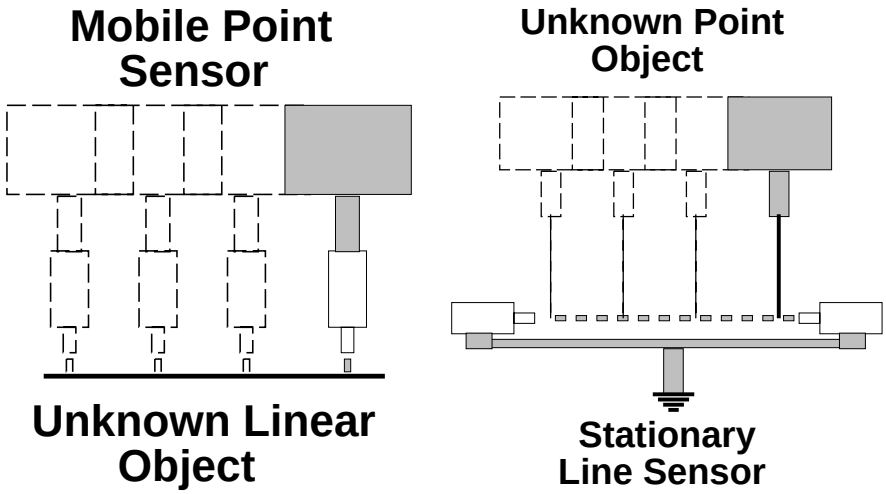
10.1.2 First Duality

Note the duality between the calibration of stationary point sensors and mobile line sensors (refer Figures 10.6(a) and 10.6(b)), due to the fact that both of these tasks involve solving the same problem: determining the position of the line and point given robot configurations (X_R, Y_R, θ_R) where the line is attached to the robot end effector and the point is stationary. The calibration of stationary line sensors and mobile point sensors are also dual problems (refer Figures 10.5(a) and 10.5(b)). We parameterize the point position by (X, Y) and the line by (R, θ).

We are going to solve for the optimal estimates of the point (X, Y) and the line (R, θ), but in each case, only two of the parameters are relevant: (X, Y) for point sensors and (R, θ) for line sensors.

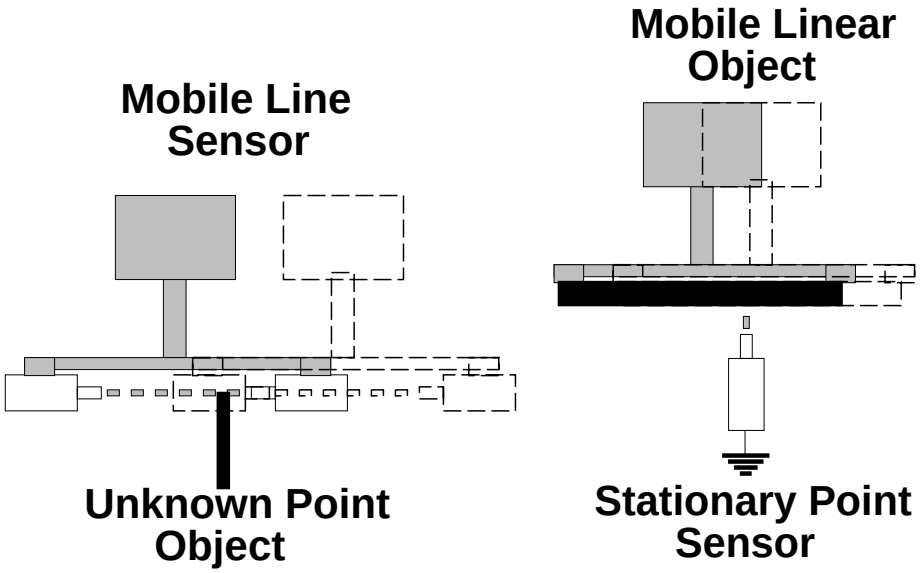
10.1.3 Least Squares Approach

Because of errors from various sources, we use least squares methods to estimate the sensor and object parameters (X, Y, R, θ) from the robot configurations (X_R, Y_R, θ_R). The task, therefore, is to determine the (X, Y, R, θ) parameters which minimize the sum



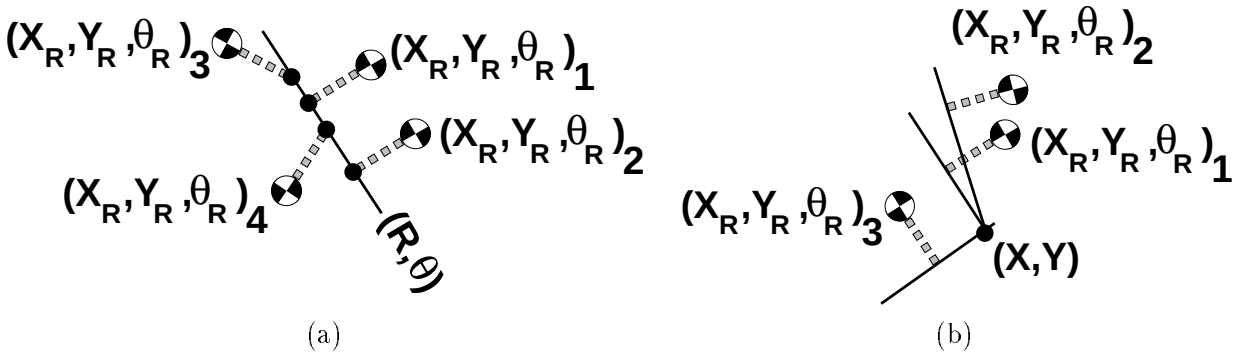
(a) (b)

Figure 10.5: Side View: (a) calibrating the mobile point sensor by maintaining contact with a linear object, and (b) the dual problem of calibrating a stationary line sensor by maintaining contact with a point object



(a) (b)

Figure 10.6: Side View: (a) calibrating the mobile line sensor by maintaining contact with a point object, and (b) the dual problem of calibrating a stationary point sensor by maintaining contact with a linear object



(a) (b)

Figure 10.7: Top Views: The task is to localize a line and a point given configurations $\{(X_R, Y_R, \theta_R)\}$ for which the point intersects the line. In (a), the point is attached to the end effector, and in (b), the line is attached to the end effector. The crosshatched circle represents the robot's center of rotation.

squared discrepancy between the predicted robot configurations and the actual configurations. For example, if a line is attached to the end effector, we want to find (X, Y, R, θ) such that we minimize the sum squared error between lines (R, θ) transformed by the robot configuration (X_R, Y_R, θ_R) and the stationary point (X, Y). We rely on global algebraic methods to solve this problem, because it is non-linear.

10.1.4 Second Duality

The second duality involved in calibration is that both pairs of problems reduce to the same least squares formulation. Although in one case, the line is transformed by the robot, and in the other case the point is transformed by the robot, both problems arrive at the same type of expression to be minimized, so that we only need a single numerical solver routine. This can be seen by considering the relationship between the reference frame attached to the robot and the world reference frame: the transformation between the world reference frame and the attached reference frame is exactly the inverse of the transformation between the attached reference and the world reference frame. Thereby, the pairs of dual problems are duals themselves. Similar dualities have been noted previously for convex polygons by Dobkin *et al.* between point scanning and line probing [DEY86]. Consequently, we only needed to implement a single numerical routine to solve for the sensor position from the configurations. After implementing the numerical solver routine for the case where the point is attached to the end effector, we are finished, since we can reduce other case to this case by making a suitable transformation of the robot

configurations. This transformation involves substituting the coordinates corresponding to the inverse transformation $(\Leftrightarrow X_R \cos(\theta_R) \Leftrightarrow Y_R \sin(\theta_R), X_R \sin(\theta_R) \Leftrightarrow Y_R \cos(\theta_R), \Leftrightarrow \theta_R)$ for the robot configuration (X_R, Y_R, θ_R).

10.1.5 Experiment and Results

The experiment consisted of calibrating different types of sensors, and measuring the accuracy of the calibration via self validation techniques. We graphed the sum squared error as functions of the size of the calibration set. We found the sensor calibration techniques accurate to 0.025 mm, which is the positioning accuracy of our Robotworld manipulator.

10.1.6 Overview

This chapter continues as follows: in section 2, we describe the theoretical framework, including the Robotworld model, the model of rotating the point or line attached to the end effector, and resultants, the algebraic method for solving systems of equations. In section 3, we detail the case where the point is attached to the end effector, and in section 4, we detail the case where the line is attached to the end effector. In section 5, we discuss a method for utilizing calibration data corresponding to multiple sensors when a single object is used. In section 6, we describe the experiments and results. We conclude by highlighting the results.

10.2 Theoretical Framework

In this section, we describe the theoretical framework, including the Robotworld system, and the model of rotating the point or line attached to the end effector. We describe the procedures for calibrating crossbeam and scanning beam sensors. Then we discuss the disadvantages of estimating line parameters (R, θ) by line fitting the (X_R, Y_R) positions. Next, we present an overview of resultants, an algebraic method for solving systems of equations.

10.2.1 Robotworld

The Robotworld system (refer Figure 10.8) running SIOMS [Nic94] was used to evaluate sensor and calibration techniques [ATT67]. Robotworld is tailored to vertical assembly tasks which constitute up to 60% of all assembly operations [NW78]. Robotworld consists of multiple four degree (x, y, z, θ) of freedom modules which cooperate or work individually in a confined workspace. Each module consists of a base which moves in the (x, y) directions actuated by Sawyer motors, and an end effector arm which extends downward from the base and rotates and translates about a vertical axis. The modules can move at speeds of up to one meter per second, and are repeatably positionable to 0.025 mm and $\frac{1}{150}^{o}$. Robotworld modules can be kinematically calibrated to extremely high precision (0.025 mm) (Chapter 11,[Wal95a]). Since the precision of this calibration technique only depends upon the local precision of the manipulator, SCARA-type Adept robots would also be appropriate.

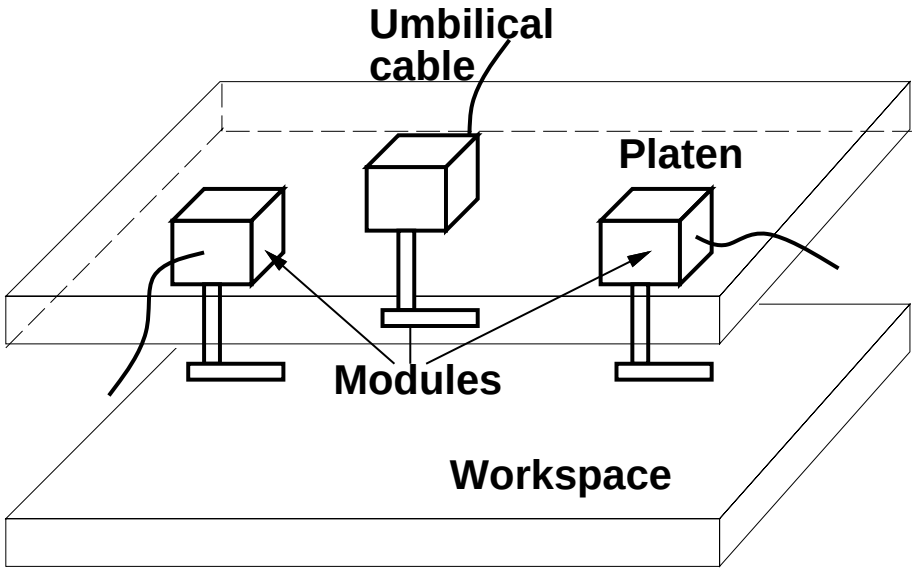


Figure 10.8: The Robotworld system

10.2.2 Model of Point Attached to End Effector

The robot configuration (X_R, Y_R, θ_R) corresponds to rotating a point (X, Y) attached to the end effector by θ_R and then translating the point by (X_R, Y_R) (refer Fig-

ure 10.9).

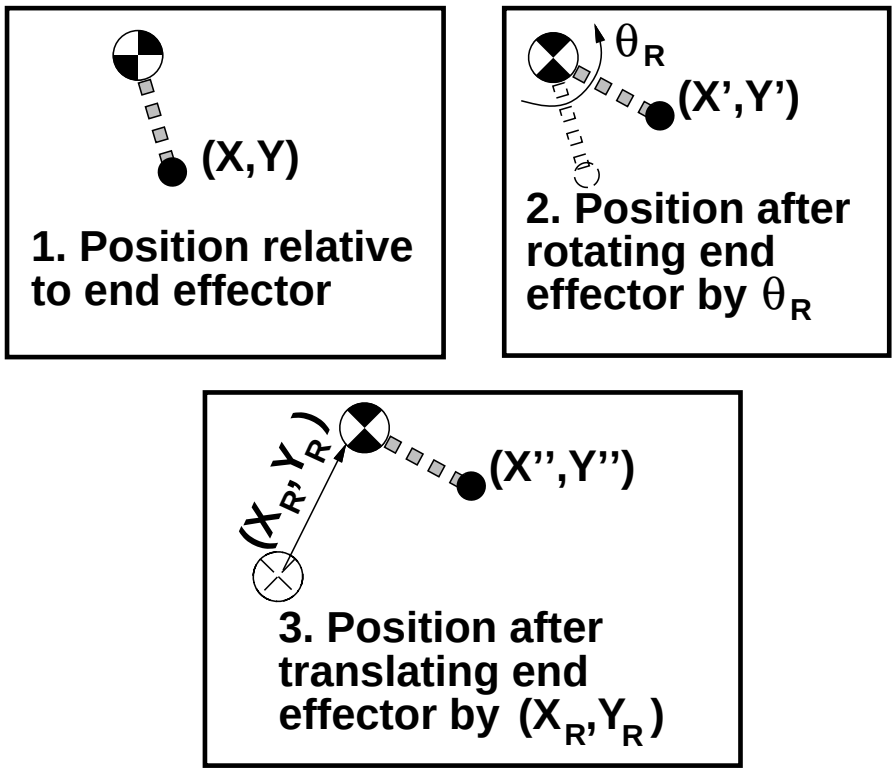


Figure 10.9: Model of point attached to end effector.

10.2.3 Model of Line Attached to End Effector

The robot configuration (X_R, Y_R, θ_R) corresponds to rotating a line (R, θ) attached to the end effector by θ_R and then translating the point by (X_R, Y_R) (refer Figure 10.10).

10.2.4 Calibrating Stationary Line Probe Sensors and Mobile Point Scanning Sensors

Calibrating stationary line probe sensors involves affixing a calibration peg to the end effector and then passing the peg through the sensor along parallel paths (refer Figure 10.11). Calibrating a mobile scanning sensor involves placing an edge object in the workspace and along various paths.

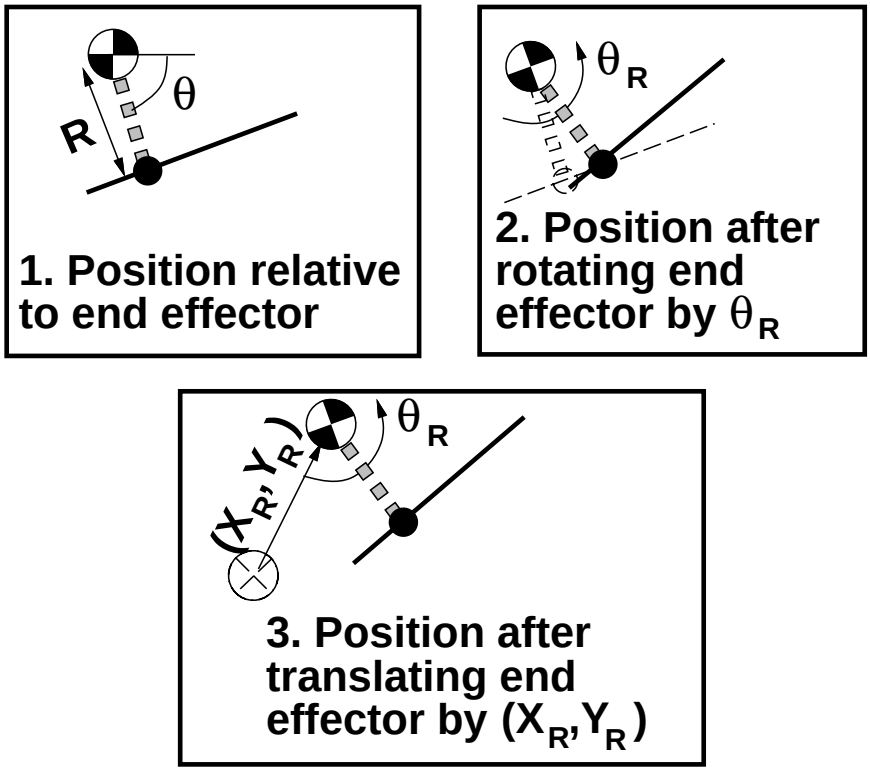


Figure 10.10: Model of line attached to end effector.

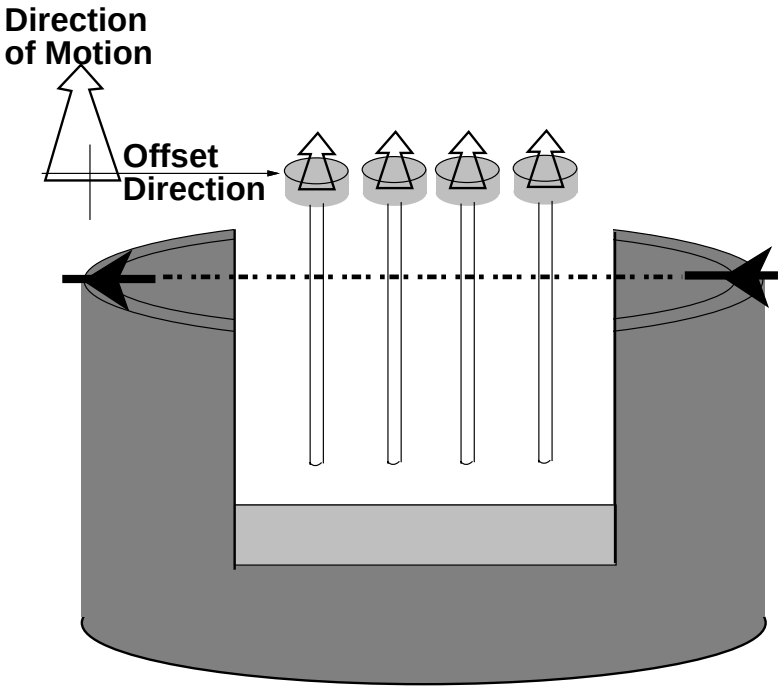


Figure 10.11: A $\frac{1}{4}''$ diameter calibrating peg is passed through the stationary crossbeam sensor to localize the line probe beams.

Estimating Line Parameters (R, θ) by Fitting A Line To Positions (X_R, Y_R)

The line parameters (R, θ) can be found by fitting a line through the robot positions (X_R, Y_R) (refer Figure 10.12). This approach is valid if the data set includes diametrically opposite θ_R robot configuration orientations, because, then, to a first order approximation, the effect of the peg's offset (X, Y) is cancelled out. We can then compute the point position (X, Y) using the line estimate, by solving the linear least squares system described in section 10.5.1.

This approach is well suited to cases where the (X, Y) offset position is relatively insignificant, but it has many problems when the (X, Y) offset position is relatively large. Furthermore, it is not intuitively obvious how to compute the optimal point estimate geometrically without resorting to the algebraic approach we describe in section 10.5.1.

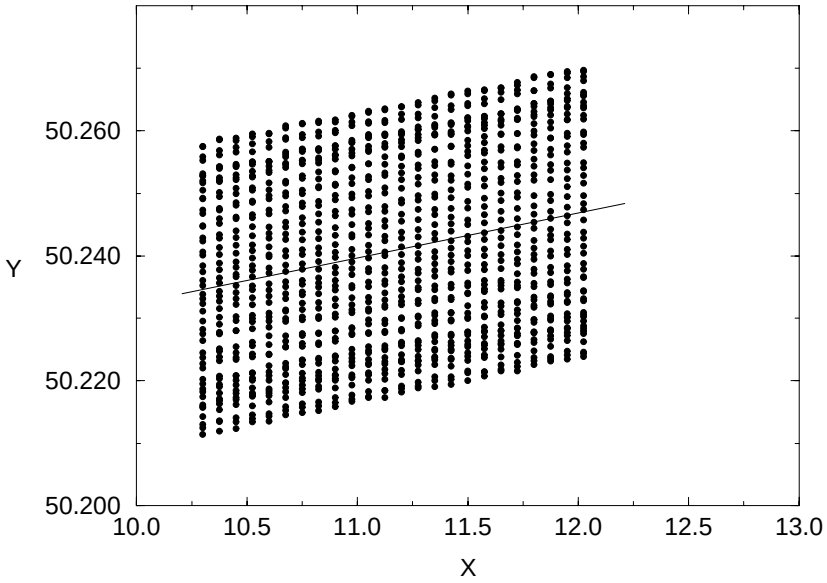


Figure 10.12: The beam orientation can be determined by fitting a line to the robot configurations (X_R, Y_R).

10.2.5 Calibrating Mobile Line Probe Sensors and Stationary Point Scanning Sensors

Calibrating mobile line probe sensors involves fixing a calibration peg in the workspace, and passing the mobile line probe sensor over the peg at various orientations (refer Figure 10.13). Calibrating a stationary scanning sensor involves affixing an edge object to the end effector and passing the object over the stationary scanning sensor along various paths.

We solve the least squares problem by finding the global minimum of an algebraic function in the same way we solved for the optimal pose estimate in Chapter 4.

10.3 Determining (X, Y, R, θ) When The Point Is Attached to End Effector

In this section, we describe how we estimate the point and line positions (X, Y, R, θ) from the robot configurations (X_R, Y_R, θ_R) when the point is attached to the end effector and the line is fixed in the workspace. This corresponds to calibrating stationary line probe sensors or mobile point scanning sensors.

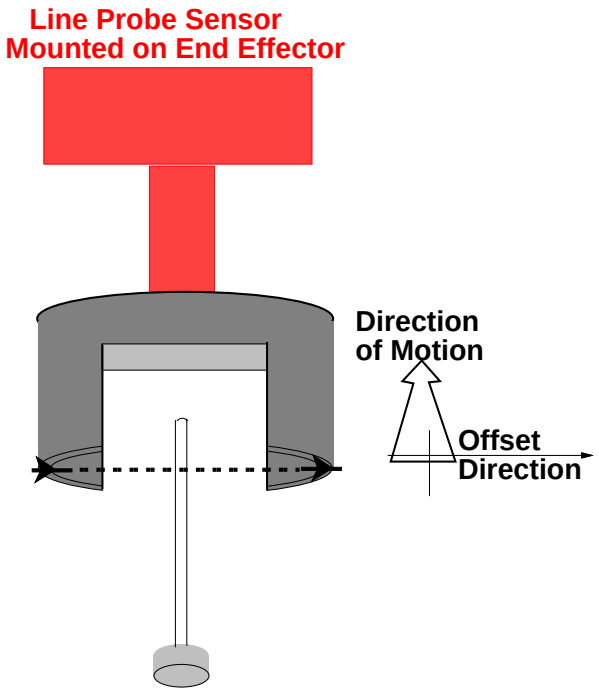


Figure 10.13: A cylindrical column fixed in the workspace is used to calibrate the mobile crossbeam sensor.

We use the same basic approach as was described in Chapter 4: formualte a sum squared error function algebraically, and solve for the global minimum using exact global techniques. Given the robot configuration (X_R, Y_R, θ_R), equation (10.1), Figure 10.14 describes the error function associated with the parameters peg offset (X, Y) and line position (θ, R). The sum squared error function $E_\Sigma(X, Y, R, \theta)$ is given in equation (10.2). We are going to find the global minimum of $E_\Sigma(X, Y, R, \theta)$ by finding all of the local extrema by solving for the (X, Y, R, θ) configurations for which all of the partial derivatives are zero $(\nabla \cdot E_\Sigma(X, Y, R, \theta) = 0)$. We use the resultant method of elimination to solve for one variable at a time, starting with θ; after solving for θ, the remaining configuration parameters can be solved using linear methods.

We use the change of variables $t = \tan(\frac{\theta}{2})$ because resultant techniques are only applicable to algebraic systems of equations. We use a lazy evaluation approach in which we construct a symbolic resultant offline, and then instantiate the coefficients for each particular system. For this reason, we formulate $E_\Sigma(X, Y, R, \theta)$ in terms of sums of separate coeffcients: $\Sigma\cos(\theta_R)$, $\Sigma\sin(\theta_R)\cos(\theta_R)$; thereby, all of the symbolic operations can be performed on the

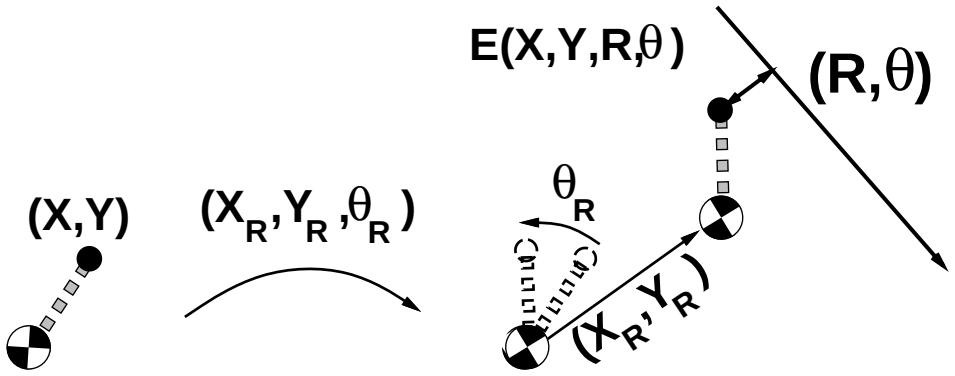


Figure 10.14: The error function $E(X, Y, R, \theta)$ for a given robot configuration (X_R, Y_R, θ_R)

generic expression, and we can instantiate the coefficients at runtime.

$$E(X,Y,R,\theta) = \begin{bmatrix} \cos(\theta_R) & -\sin(\theta_R) & X_R \\ \sin(\theta_R) & \cos(\theta_R) & Y_R \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ -R \end{bmatrix} \tag{10.1}$$

$$E_\Sigma(X,Y,R,\theta) = \sum E^2(X,Y,R,\theta) \tag{10.2}$$

$$\begin{aligned} E_\Sigma(X,Y,(1+t^2)^2,R) &= \\ &\Sigma\cos(\theta_R)(2RX + 4RtY + 4Rt^3Y - 2Rt^4) \\ &+\Sigma\sin(\theta_R)\cos(\theta_R)(-2(1-6t^2+t^4)XY + 4(t-t^3)(X^2-Y^2)) \\ &+\Sigma\cos(\theta_R)X_R(2X + 4tY - 4t^2X - 4t^3Y + 2t^4X) \\ &+\Sigma\cos(\theta_R)Y_R(4tX + 8t^2Y - 4t^3X) \\ &+\Sigma\sin(\theta_R)(-2RY + 4RtX + 4Rt^3X + 2Rt^4Y) \\ &+\Sigma\sin(\theta_R)X_R(-2Y + 4tX + 4t^2Y - 4t^3X - 2t^4Y) \\ &+\Sigma\sin(\theta_R)Y_R(-4tY + 8t^2Y + 4t^3Y) \\ &+\Sigma\cos^2(\theta_R)((1-t^2)^2X^2 + 4tXY + 4tY^2 - 4t^3XY) \\ &+\Sigma\sin^2(\theta_R)((1-t^2)^2Y^2 - 4tXY + 4tX^2 + 4t^3XY) \\ &+\Sigma 1(R^2(1+t^2)^2 + \Sigma X_R((1-t^4)2R) + \Sigma XR^2((1-t^2)^2) \\ &+\Sigma Y_R(t+t^3)4R + \Sigma X_RY_R(t-t^3)4R + \Sigma YR^2(4t^2) \end{aligned} \tag{10.3}$$

The global minimum is computed by finding all of the local extrema, and returning the extrema corresponding to the minimum error. The local extrema correspond to configu-

rations (X, Y, R, θ) for which $\frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial X} = \frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial Y} = \frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial \theta} = \frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial R} = 0$. This method is guaranteed to extract the global minimum, because all of the extrema are found.

First, we compute the partial derivatives of $E_\Sigma(X, Y, R, \theta)$ in terms of the symbolic coefficients. To arrive at algebraic expressions, we define $G_X(t), G_Y(t), G_t(t), G_R(t)$ to be the products of $\frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial X}, \frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial Y}, \frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial \theta}, \frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial R}$ multiplied by the denominator of the rational expressions, $(1+t^2)^2$. The partial derivative $\frac{\partial E_\Sigma(X,Y,R,\theta)}{\partial \theta}$ takes the change of variables into account (refer equation (10.4)). The resultant $M(t)$ involves the monomial components of $G_X(t), G_Y(t), G_R(t), G_t(t)$, $G_{XX}(t), \ldots$ following the convention that the first letter subscript refers to the partial derivative, and the remaining letters refer to the monomial coefficient; *i.e.*, $G_{XY}(t)$ corresponds to the coefficient corresponding to the Y monomial in $G_X(t)$, and $G_{tXX}(t)$ corresponds to the coefficient corresponding to the X^2 monomial in $G_t(t)$.

$$\frac{\partial \frac{E_\Sigma(X,Y,t,R)}{(1+t^2)^2}}{\partial t} = \frac{(1+t^2)\frac{\partial E_\Sigma(X,Y,t,R)}{\partial t} - 4tE_\Sigma(X,Y,t,R)}{(1+t^2)^3} \tag{10.4}$$

$$\begin{aligned}
\frac{\partial E_\Sigma}{\partial X} &= G_{XX}(t)X + G_{XY}(t)Y + G_{XR}(t)R + G_{X1}(t) \\
\frac{\partial E_\Sigma}{\partial Y} &= G_{YX}(t)X + G_{YY}(t)Y + G_{YR}(t)R + G_{Y1}(t) \\
\frac{\partial E_\Sigma}{\partial R} &= G_{RX}(t)X + G_{RY}(t)Y + G_{RR}(t)R + G_{R1}(t) \\
\frac{\partial E_\Sigma}{\partial \theta} &= G_{tXX}(t)X^2 + G_{tYY}(t)Y^2 + G_{tXY}(t)XY + \\
&\quad G_{tXR}(t)XR + G_{tYR}(t)YR + G_{tX}(t)X + \\
&\quad G_{tY}(t)Y + G_{tR}(t)R + G_{t1}(t)
\end{aligned}$$

The Macaulay construction of the resultant $M(t)$ (which includes extraneous factors) of these partial derivatives is given in equation (10.5); the (t) was left out of all of the $G_{XX}(t), \ldots$ for space reasons.

$$M(t) = \tag{10.5}$$

$$\begin{bmatrix}
G_{XX} & G_{XY} & G_{XR} & G_{X1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & G_{XX} & 0 & 0 & G_{XY} & G_{XR} & G_{X1} & 0 & 0 & 0 \\
0 & 0 & G_{XX} & 0 & 0 & G_{XY} & 0 & G_{XR} & G_{X1} & 0 \\
0 & 0 & 0 & G_{XX} & 0 & 0 & G_{XY} & 0 & G_{XR} & G_{X1} \\
0 & G_{YX} & 0 & 0 & G_{YY} & G_{YR} & G_{Y1} & 0 & 0 & 0 \\
0 & 0 & G_{YX} & 0 & 0 & G_{YY} & 0 & G_{YR} & G_{Y1} & 0 \\
0 & 0 & 0 & G_{YX} & 0 & 0 & G_{YY} & 0 & G_{YR} & G_{Y1} \\
0 & 0 & G_{RX} & 0 & 0 & G_{RY} & 0 & G_{RR} & G_{R1} & 0 \\
0 & 0 & 0 & G_{RX} & 0 & 0 & G_{RY} & 0 & G_{RR} & G_{R1} \\
G_{tXX} & G_{tXY} & G_{tXR} & G_{tX} & G_{tYY} & G_{tYR} & G_{tY} & 0 & G_{tR} & G_{t1}
\end{bmatrix}$$

10.3.1 Solving the Matrix Polynomial $|M(t)| = 0$

$M(t)$ was obtained (equation (10.5)) by eliminating X, Y and R from the equations corresponding to the partial derivatives of $E_\Sigma(X, Y, R, t)$. In this section, we highlight a numerically accurate and efficient algorithm to compute the roots of $|M(t)| = 0$. All of the t coordinates of the common solutions of the algebraic equations are roots of $|M(t)| = 0$.

Solving $|M(t)| = 0$ is reduced to determining the eigenvalues of the block companion matrix E (refer equation (10.6)). The construction of the block companion matrix E is highlighted in equation (10.7). It involves matrix inversion and matrix products. We assume that the leading matrix, M_4, is numerically well-conditioned, and there are also methods to overcome ill-conditioned matrices [Man92]. The eigenvalues of E are exactly the roots of the equations $|M(t)| = 0$ [Man92]. Solving for the roots of $M(t)$ in this manner is numerically better conditioned than solving the resultant polynomial formed by expanding the resultant matrix.

$$E = \begin{bmatrix}
0 & I & 0 & 0 \\
0 & 0 & I & 0 \\
0 & 0 & 0 & I \\
M_4^{-1}M_0 & M_4^{-1}M_1 & M_4^{-1}M_2 & M_4^{-1}M_3
\end{bmatrix} \tag{10.6}$$

$$M(t) = M_4t^4 + M_3t^3 + M_2t^2 + M_1t + M_0 \tag{10.7}$$

10.4 Determining (X, Y, R, θ) When Line Attached to End Effector

In this section, we detail the computation involved in localizing a point and a line for the case where the line is attached to the end effector, and the point is fixed in

the workspace. This corresponds to calibrating mobile line probe sensors and stationary scanning point sensors.

We can reduce this case, the case where the line is attached to the end effector, to the other case, the case where the point is attached to the end effector, by making a suitable transformation of the robot configurations. This transformation involves substituting the coordinates corresponding to the inverse transformation ($\Leftrightarrow X_R \cos(\theta_R) \Leftrightarrow Y_R \sin(\theta_R)$, $X_R \sin(\theta_R) \Leftrightarrow Y_R \cos(\theta_R)$, $\Leftrightarrow \theta_R$) for the robot configuration (X_R, Y_R, θ_R). This can be seen by considering the relationship between the reference frame attached to the robot and the world reference frame: the transformation between the world reference frame and the attached reference frame is exactly the inverse of the transformation between the attached reference and the world reference frame.

10.5 Utilizing Calibration Data From Multiple Sensors For The Same Object

Up to this point, we have discussed techniques for calibrating a single sensor using a single object. Calibrating multiple sensors with the same object provides more information because the object's parameters ((X, Y) for a point object, or (R, θ) for a line object) remain constant throughout all of the calibration experiments. Sensor position estimates are improved by incorporating this constraint into the numerical solver.

Hill climbing methods can be used in conjunction with exact approaches to incorporate these constraints. First we estimate the unknown object's position for each sensor individually to provide a starting configuration. Then we rely on hill climbing methods to minimize the total error for all the sensors and robot configurations. It turns out that it is much easier to solve for the sensor's position when the object's position is known.

10.5.1 Solving for X and Y Given R and θ

Given the position estimates for a line (R, θ), the (X, Y) sensor position can be computed by solving the linear system of equations (10.8,10.9) corresponding to $\frac{\partial E_{\Sigma(X,Y,R,\theta)}}{\partial X}|_{R,\theta}$ $= \frac{\partial E_{\Sigma(X,Y,R,\theta)}}{\partial Y}|_{R,\theta} = 0$. The simultaneous solution is found using linear methods.

$$G_{XX}X + G_{XY}Y \quad = \quad \Leftrightarrow G_{XR} \Leftrightarrow G_{X1} \tag{10.8}$$

$$G_{YX}X + G_{YY}Y = \Leftrightarrow G_{YR} \Leftrightarrow G_{Y1} \tag{10.9}$$

10.5.2 Solving for θ and R Given X and Y

Given the position estimates for a point (X, Y), the (R, θ) sensor position can be computed using resultant techniques (refer equations (10.10,10.11) corresponding to $\frac{\partial E_{\Sigma(X,Y,R,\theta)}}{\partial R}|_{X,Y} = \frac{\partial E_{\Sigma(X,Y,R,\theta)}}{\partial \theta}|_{X,Y} = 0$. The resultant $M(t)$ corresponding to this system of equations is given in equation (10.12). The t roots of the resultant correspond to those values for which $|M(t)| = 0$. Due to the nature of the individual components, the symbolic determininant is divisible by $(1 + t^2)^2$ producing a quartic algebraic expression which can be solved exactly using numerical methods.

$$G_{tR}(t)R + \underbrace{G_{t1}(t) + G_{tX}(t) + G_{tY}(t)}_{R\ independent} = 0 \tag{10.10}$$

$$G_{RR}(t)R + \underbrace{G_{R1}(t) + G_{RX}(t) + G_{RY}(t)}_{R\ independent} = 0 \tag{10.11}$$

$$M(t) = \begin{bmatrix} G_{tR}(t) & G_{t1}(t) + G_{tX}(t) + G_{tY}(t) \\ G_{RR}(t) & G_{R1}(t) + G_{RX}(t) + G_{RY}(t) \end{bmatrix} \tag{10.12}$$

10.6 Experiments and Results

In this section, we describe the experiments and results used to measure this technique's accuracy.

10.6.1 Experiment

The experiment consisted of calibrating various sensors, and utilizing self validation techniques to measure the quality of these estimates. Using the Robotworld manipulator, we collected 1,152 robot position measurements for six line probe sensors by passing the object along 48 different paths at 24 different orientations. We then computed the sensor and object positions from this data. Next, we performed self validation techniques by estimating the sensor and object parameters from a randomly chosen sample of the experimental data set, and then computing the standard deviations of the error function $E_\Sigma(X, Y, R, \theta)$ (equation (10.2)) corresponding to the entire data set. For each sample size, we chose 20 different samples.

Sensor	X (mm)	Y (mm)	θ (rad)	R (mm)	σ (mm)
Top Crossbeam 0^o	-0.01533	0.01339	-1.5641	-50.164	0.0022
Top Crossbeam $\Leftrightarrow 45^o$	-0.01608	0.01433	0.7933	43.732	0.0161
Top Crossbeam 45^o	-0.01548	0.01408	2.3624	27.287	0.0087
Bottom Crossbeam 0^o	-0.01900	0.01515	-1.5646	-50.174	0.0050
Bottom Crossbeam $\Leftrightarrow 45^o$	-0.01931	0.01589	-2.3480	-43.733	0.0095
Bottom Crossbeam $45^o 6$	-0.01858	0.01551	2.3595	27.410	0.0033

Table 10.1: The peg positions (X, Y) and line probe sensor parameters (θ, R) and standard deviation estiamtes determined from a data set of 1150 robot configurations.

Sensor	X (mm)	Y (mm)	θ (rad)	R (mm)	σ (mm)
Scanning Beam 1	41.1779	50.1956	1.5419	0.02877	0.0033
Scanning Beam 2	44.9795	50.0606	1.5403	0.02801	0.0060
Scanning Beam 3	38.6875	50.2176	-1.6022	-0.02843	0.0034
Scanning Beam 4	43.7160	50.1004	1.5392	0.02878	0.0026
Scanning Beam 5	42.3935	50.1888	1.5410	0.02870	0.0023
Scanning Beam 6	39.8848	50.1863	1.5412	0.02959	0.0032

Table 10.2: The scanning sensor positions (X, Y), line object parameters (θ, R), and standard deviation estimates σ determined from a data set of 152 robot configurations.

10.6.2 Results

Table 10.1 includes the computed crossbeam line probe (R, θ) parameters computed from 1150 calibration experiments. The standard deviation estimates allow us to quantify the sensor's precision. The standard deviation error σ corresponding to Top Crossbeam 0^o is 0.0022 mm, and the standard deviation error σ corresponding to Top Crossbeam $\Leftrightarrow 45^o$ is 0.0161 mm.

Table 10.2 includes the computed stationary scanning beam positions (X, Y) computed from 152 calibration experiments. The standard deviation error σ corresponding to Scanning Beam 2 is 0.0060 mm, and the standard deviation error σ corresponding to Scanning Beam 5 is 0.0023 mm.

Figures 10.15, 10.16, and 10.17 depict the standard deviation of the sensor pose estimates as a function of the sample size. We used a self validation technique in which we estimated the sensor and object parameters using a randomly selected a sample set of a certain size, and then computed the error function $E_\Sigma(X, Y, R, \theta)$ for those parameters with respect to the entire dataset.

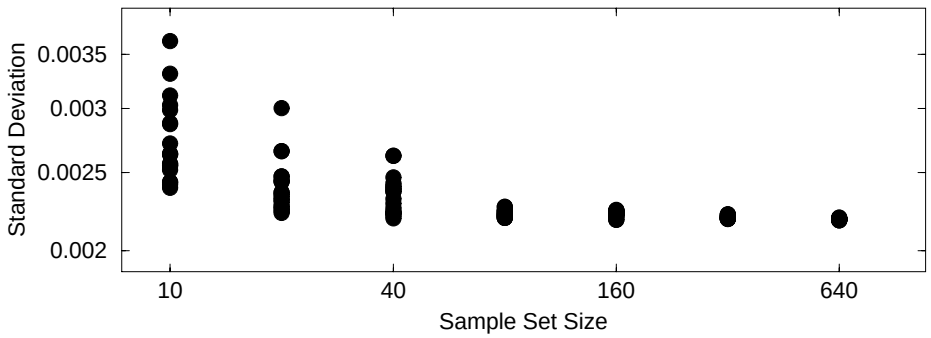


Figure 10.15: Standard deviation estimates for Crossbeam Top 0^o as a function of sample set size as measured via self validation

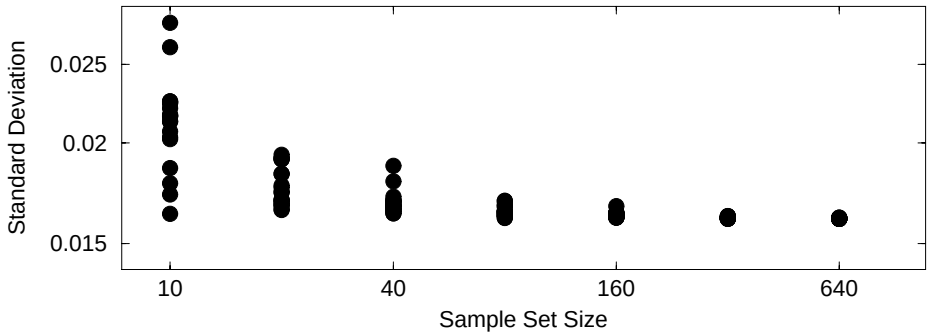


Figure 10.16: Standard deviation estimates for Crossbeam Top $\Leftrightarrow 45^o$ as a function of sample set size as measured via self validation

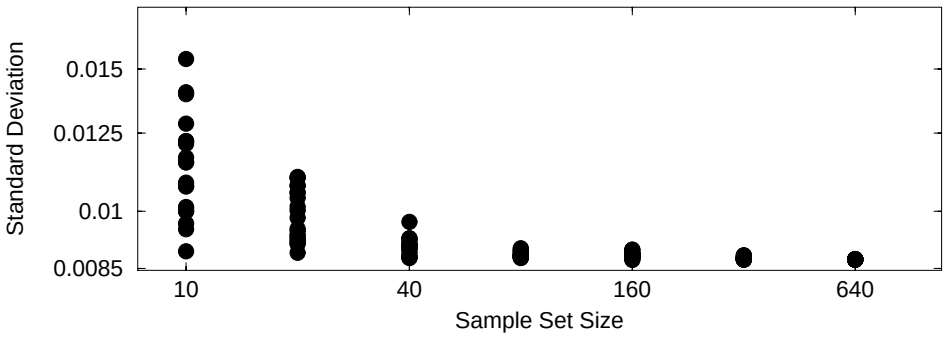


Figure 10.17: Standard deviation estimates for Crossbeam Top 45^o as a function of sample set size as measured via self validation

10.7 Conclusion

Recently, many researchers have demonstrated that simple binary beam sensors can perform a wide range of industrial tasks. In this chapter, we detailed techniques for calibrating line probing and point scanning sensors from robot configurations in which the sensor registered an object. These techniques were applicable to both stationary and mobile sensors. We observed two dualities inherent in calibration. Calibrating a mobile point sensor and calibrating a stationary line sensor are dual problems becuase they both correspond to localizing a line and point given data of the form: robot configurations (X_R, Y_R, θ_R) correspond to a stationary line contacting a point attached to an end effector; for similar reasons, calibrating a stationary point sensor and calibrating a mobile line sensor are dual problems. The second duality inherent in calibration is that both problems involve the same numerical computation to solve for the sensor position as well as the object position.

Chapter 11

Calibration of Four Degree of Freedom Robotworld Modules

Abstract

Although many researchers focus on anthropomorphic six degree of freedom robots, they are more complicated, less accurate, and more difficult to calibrate, than four degree of freedom robots. Four degree of freedom robots constitute a significant fraction of industrial robots because specilized four degree of freedom robots can accurately perform vertical assembly operations, which consitute 60% of all manufacturing operations. Still, most calibration research has focused on anthropomorphic robots. In this chapter we describe calibration techniques for Robotworld modules [Sch87], simple four degree of freedom (x, y, z, θ) *manipulators designed for vertical assembly operations which combine high performance and simple calibration. Robotworld modules are implicitly coordinated in* x *and* y *to approximately* 0.025 *mm due to the fact that they all move with respect to the same platen; therefore, we focus on the task of modeling the end effector arm's axis of rotation and axis of translation. These axes are estimated by localizing a calibration peg using multiple crossbeam sensors at various heights. Cross validation results suggest that calibration provides absolute accuracy of* 0.025 *millimeters.*

11.1 Introduction

Before the personal computer revolution, many believed that computers were too remote and too abstract for the average person. The main factors underlying the revolution were decreasing cost and increasing ease of use. We believe that robots need to be simpler to install, program, and operate before they become more prevalent in industry. Robots can provide high performance, especially if multiple robots cooperate to perform a task, but cooperation requires coordination, the ability of a manipulaor to determine the relative positions of other manipulators [Maz88; MA89]. An effective coordination technique would partially solve all of these tasks. Coordination is a prerequisite for utilizing offline programming, and is the foundation of device integration [Cra92].

Kinematic calibration, the modeling of a device's kinematic parameters, is one method of achieving device coordination. Although anthropomorphic robots can be calibrated, the resulting models do not provide accurate enough prediction (0.3 mm); this may be due to the fact that anthropomorphic robots are mechanically complex, making them difficult to model; therefore, anthropomorphic robots can be too imprecise for some industrial applications [MP89b; BM89b; Sur90], although other applications still require their use. In contrast, Robotworld modules [ATT67; Sch87], which have fewer degrees of freedom and are tailored to vertical assembly operations which constitute up to 60% of all assembly operations [NW78], are well suited to kinematic calibration.

In this paper, we present a calibration technique, and we show (using self validation) that as 40 measurements can reduce the standard deviation error estimate to 0.02 mm. This introduction is organized as follows: first we describe the Robotworld system, then we describe the calibration technique, and then we discuss the previous work.

11.1.1 Robotworld

In Robotworld (Chapter 10.2.1), multiple four degree of freedom (x, y, z, θ) manipulators (accurate to 0.025 mm) work either separately or cooperatively in a $1m \times 1m \times 0.1m$ workspace. Using Sawyer motors (Figure 11.1) for actuation in x and y has many advantages. Robotworld modules are fast ($1\ m/s$), repeatably accurate (0.025 mm), and all of the modules are implicity coordinated in the x and y directions because the modules all move with respect to the same platen grid. Although the modules are usually run without feedback (open loop), module x, y positions can be sensed to within 10 microns using mag-

netic sensors [IS94; ISW94] or to within 100 microns using quadrature encoders [WN93; NBWY93]. Consequently, calibration only involves characterizing the end effector arm: the z and θ degrees of freedom.

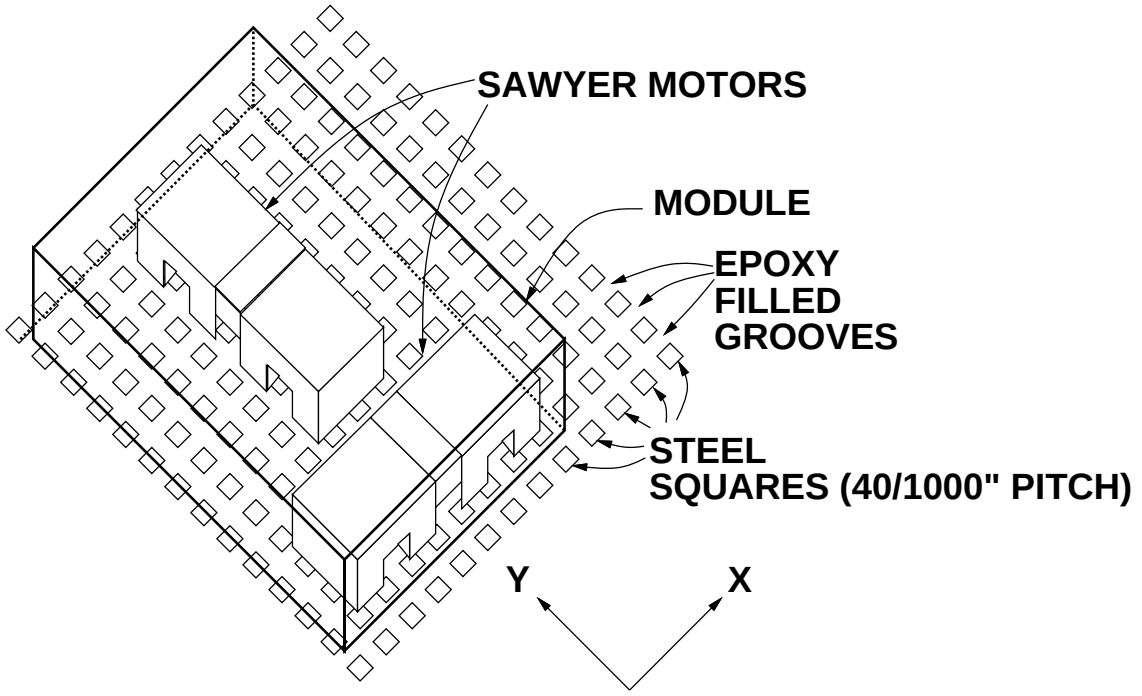


Figure 11.1: Base of a Robotworld module (depicted upside-down, with the robot above the platen)

The end effector arm extends downward from the base and rotates and translates about a vertical axis, and our model of the arm was based on the hardware mechanics (Figure 11.2). The model consists of rotation around an axis of rotation followed by translation parallel to an axis of translation. The rotation and translation joint parameters (z, θ) are assumed known to very high accuracy. The translation motor z is encoded to approximately 2000 counts per centimeter, and the rotation motor θ is encoded to approximately 150 counts per degree. Calibration only involves determining the end effector arm axes, *i.e.*, the offset and slope of the translation and rotation axes.

11.1.2 Overview of Calibration Technique

The calibration technique involves using crossbeam sensors to localize a peg attached to the end effector; the calibration peg is localized by multiple crossbeam sensors

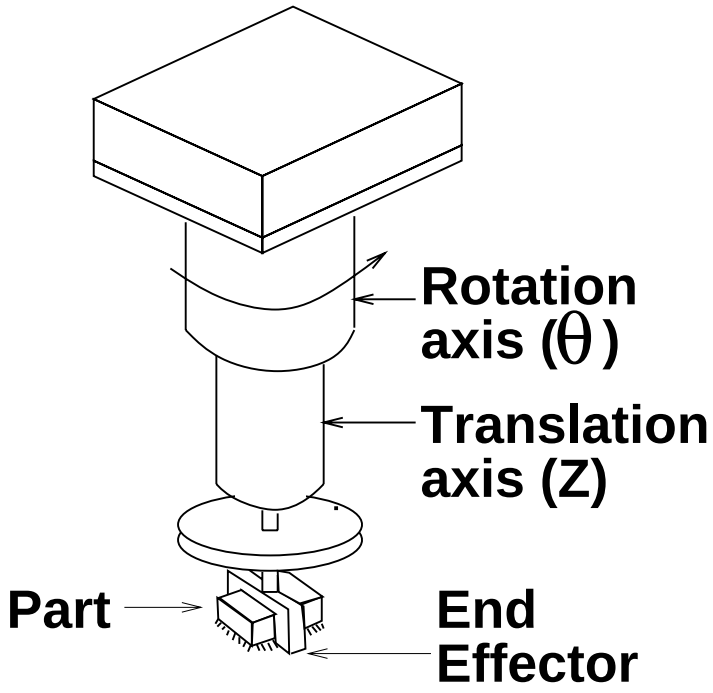


Figure 11.2: Configuration of rotation axis and translation axis of end effector arm

at different heights; the peg is localized for various z, θ robot configurations. The technique assumes only that the calibration peg is straight and cylindrical. Crossbeam sensors [WCM93a; PC94] (Figures 11.3,11.4) are composed of a set of coplanar crossed binary light beam sensors, and provide extremely high accuracy (0.025 mm) and extract pertinent information: the location of the cross section of the peg at a particular height.

There are two calibration approaches: complete information, in which a canonical point (on the end effector) is experimentally measured, and partial information, in which the some parameters of the canonical point are measured, but the user does not explicitly measure the position of the canonical point [HL93]. Approaches of the first type may involve using a coordinate measuring machine to measure the frame, whereas approaches of the second type may involve using a radial distance sensor to measure the distance from the end effector to a particular point. Our technique is of the second variety, partial information, in that we measure the position of a peg attached to the end effector, and use the peg positions to derive the kinematic model parameters.

For the purpose of simplifying the analysis, we make a linearization assumption: since the axes are so nearly vertical straight (this assumption was experimentally justified)

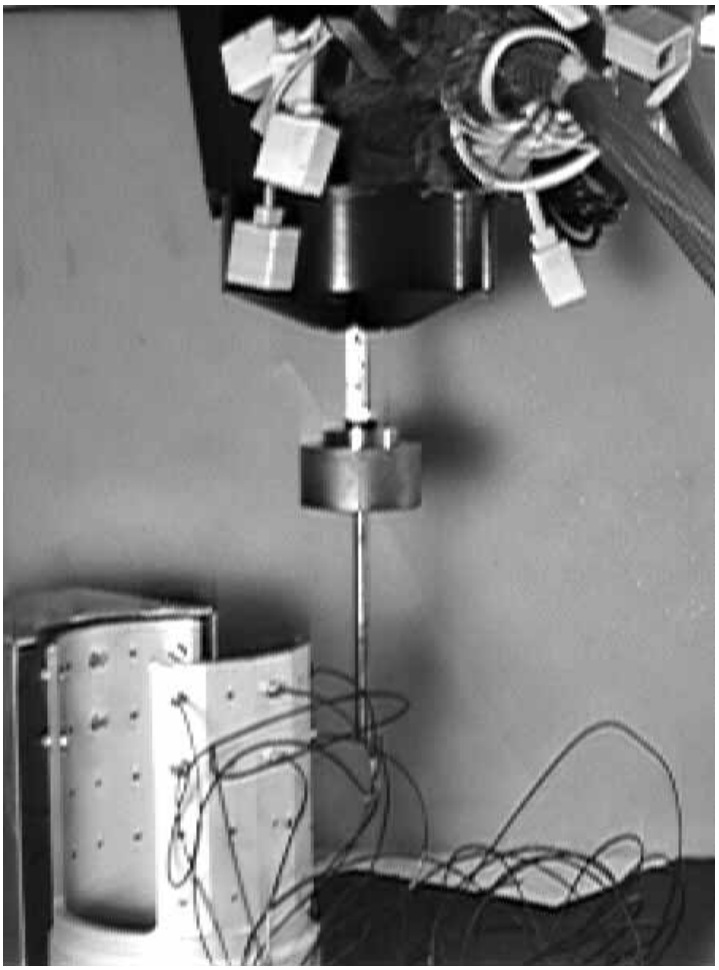


Figure 11.3: A Robotworld module, the calibration peg, and the crossbeam sensor

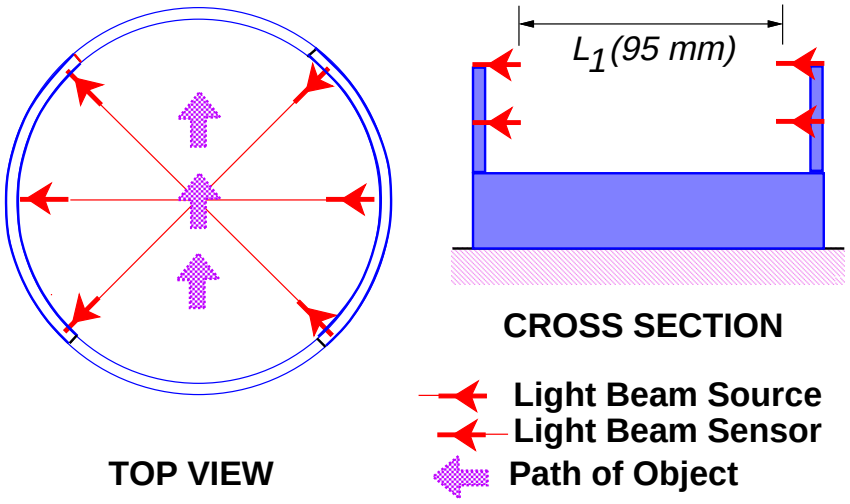


Figure 11.4: Crossbeam sensor apparatus

that the non-linear axis components are inconsequential in most regards. This assumption reduces the general problem of determining a skew line in three dimensions to the simpler problem of estimating x,y translational components as linear functions of z. Since the Robotworld axes are nearly vertical, linearization is not as large a source of error as it is for calibration of six degree of freedom robots.

We estimate the axis of rotation before we estimate the axis of translation because of the following invariant: for all z configurations, the localized peg position traces out a circle for various θ; furthermore, the circle is centered at the intersection of the axis of rotation and the horizontal crossbeam plane.

Next, we estimate the axis of translation. The rotation axis estimate is used to remove any dependence on θ by normalizing the entire dataset to $\theta = 0$. Then, the axis of translation parameters are estimated using linear methods.

Each module was precisely characterized via 960 localized peg positions; the modules were cross-calibrated to the accuracy of the Sawyer motors (0.025 mm). Cross validation techniques showed that only 80 localized peg positions provided sufficient kinematic parameter estimates to achieve absolute positional accuracy of 0.025 mm. If the end effector arm is not modeled, then the user has no way of knowing that the end effector can translate in x and y by as much as 1 mm as the z height and orientation are varied.

11.1.3 Previous Work

In this section, we review the literature and report the results to show that anthropomorphic robots are inherently difficult to model and calibrate.

Most of the previous work in kinematic calibration focused on anthropomorphic arm-based robots, and, as such, many results such as numerical techniques for six degree of freedom problems, and extremely complex error source models are untransferable and unnecessary for Robotworld manipulators. Still, some general results, such as the existence proof of uncharacterizable model parameters, are directly applicable to this problem.

However, the previous work does serve to support our hypothesis that arm-based robots are inherently difficult to model and calibrate. Suryohadiprojo [Sur90] remarks that simpler manipulators with fewer than six degrees of freedom are very attractive owing to their improved performance and simpler dynamics.

Various devices have been used to measure robot articulation, such as radial distance linear transducers [GQP93], coordinate measurement machines [MP89b; BM89a], vision-based theodolites [DP91], and lasers [PVG93; NO93]. All of these devices have the drawback that they require installing the sensor near the robot, and that the sensor may not be well suited to a manufacturing environment. Both the radial distance linear transducers and the coordinate measurement machines involve contact, and contact-based methods are susceptible to errors due to contact force. Bennett and Hollerbach showed that kinematic parameters could be identifed without any additional sensors when the actuators formed closed kinematic chains [BH91], but their approach also suffers from the problems of contact-based methods.

Goswanni *et al* utilized radial distance linear transducers to calibrate a Puma 560 robot. The simple measurement device was chosen because of its high accuracy. The discrepancies between predicted positions and actual end effector positions followed a Gaussian distribution with standard deviation $\sigma = 0.32$ mm [GQP93]. They cited random errors *such as repeatability and gear backlash*, confirming our hypothesis that many arm-based robots are excessively difficult to model accurately enough for industrial assembly applications. Of course, problems associated with gear backlash can be alleviated by using a better robot design *i.e.*, direct drive motors; this only serves to support our thesis that Robotworld is a very good manipulator.

Borm and Menq [BM89a] and Mooring and Padavala [MP89b] used coordinate

measuring machines to measure the absolute position of the Puma end effector. Borm and Menq [BM89a] discussed the importance of an observability criterion, and reported root mean squared (RMS) positional errors on the order of 0.6 mm. Mooring and Padavala investigated the effect of increasingly complex models and reported RMS positional errors on the order of 0.5 mm for the most complex model.

Camera-based, laser-based, and beam-based approaches all have the advantage of non-contact sensing. Driels and Pathre used a vision-based theolodite to measure end effector position [DP91]. Their approach involved mounting a spherical light source on the end effector and tracking the light source with a camera mounted on a controllable platform; the camera was calibrated using a coordinate measuring machine. Their calibration technique achieved repeatable positional accuracy of 0.2 mm.

A combination of lasers and CCD visual feedback has also been used to calibrate robots without contact. Prenninger *et al* describe a robust mirrored-laser based approach for determining absolute end effector position [PVG93]. Newman and Osborn describe an inexpensive and robust laser-sensor based approach which relies solely on the straightness of the beam; they also describe a heuristic for determining the kinematic parameters from this data [NO93].

Everett and Ives used a binary beam sensor (similar to ours) to calibrate industrial robots [EI93]. They extended the FIND_PEG paradigm, in which a robot is recalibrated in x, y, z by using a light beam to find a peg in the workspace, to $x, y, z, \theta, \phi, \psi$. They developed a FIND_SPHERE paradigm, which uses a set of three beam sensors to repeatably position a set of spheres, thereby recalibrating the robot to a particular reference frame. Their approach achieved repeatable *local* positioning accuracy of 0.075 mm, but they did not make any claims concerning global positioning accuracy.

In addition, Hollerbach and Lokhorst and Bennett and Hollerbach have shown that for closed kinematic chains, kinematic calibration can be achieved solely from joint angle measurements [HL93; BH91]. Such methods could be used in conjunction with Everett and Ives' FIND_SPHERE approach to accurately characterize kinematic parameters. Hollerbach and Lokhorst calibrated a three arm, six degree of freedom hand controller to 0.75 mm RMS positional error [HL93].

11.1.4 Outline

The remainder of this chapter is organized as follows: In section 2, we describe the theoretical framework, including the model of the Robotworld module. In section 3, we describe the technique for estimating the kinematic model parameters of Robotworld modules. In section 4, we describe the experiment and results, and we conclude by highlighting the contributions of this technique.

11.2 Theoretical Framework

11.2.1 Peg Position in Sensor Plane

Coplanar crossbeam sensors determine the position of an object *in the beam sensor plane*, (*i.e.*, the plane in which the three beams lie). Figure 11.5 shows a model of the calibration peg attached to the end effector of the Robotworld module. Figure 11.6 depicts various intersection positions $(I_x(z,\theta,h), I_y(z,\theta,h))$ for configurations z, θ and beam sensor plane height h.

11.2.2 Linearization Approximation

To simplify the analysis, we use linearized transformations to model the effects of the rotation and translation axes. Consider the contradictory case where the rotation axis of the end effector was 45^o off vertical; then for various orientations, the localized peg positions, the intersections of the peg with the beam plane, would correspond to a variety of points on the peg. After estimating the module's parameters model, we determined that these cases were irrelevant; incorporating this additional degree of freedom would have complicated the model, since we could no longer use the linearization assumption.

For nearly vertical axes, the discrepancies are insignificant; consider an almost vertical line of height 1, with a horizontal component of ϵ. The difference between the line's height 1 and its length $\sqrt{1+\epsilon^2}$ goes as $\frac{\epsilon^2}{2}$. In the case of axes which are skewed by 1 mm per 12 cm travel, ϵ corresponds to 1%, and $\frac{\epsilon^2}{2}$ corresponds to 0.005%, or 0.005 mm.

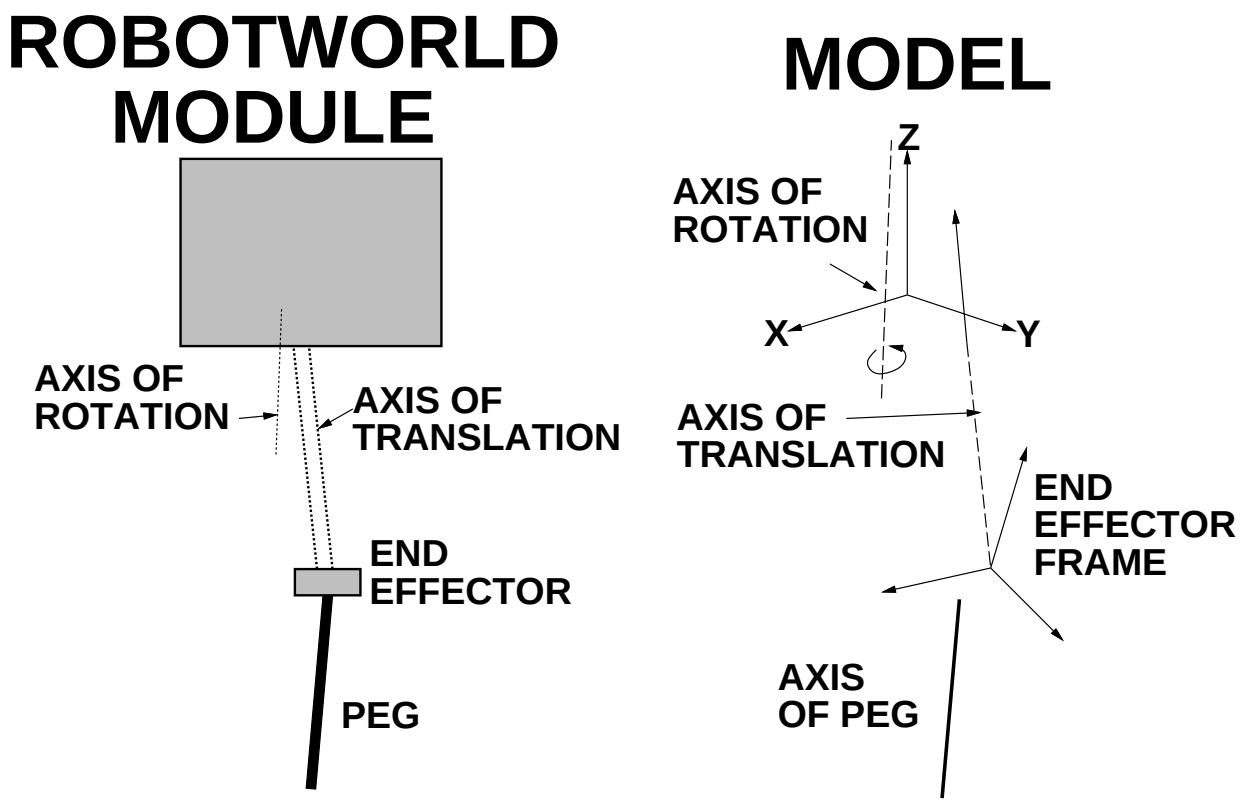


Figure 11.5: Model of calibration peg attached to end effector of Robotworld module

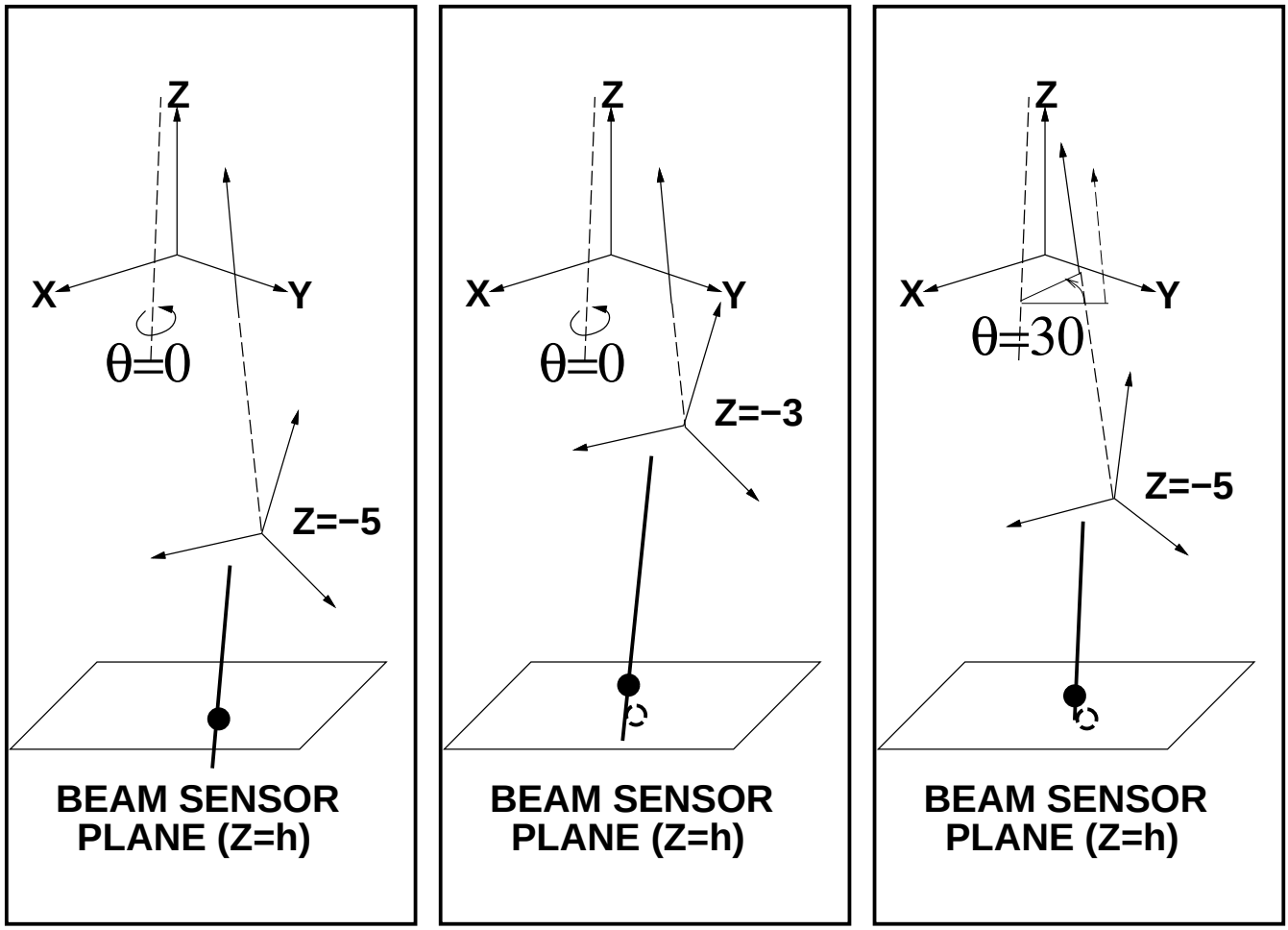


Figure 11.6: Various intersection positions $(I_x(z,\theta,h), I_y(z,\theta,h))$ for different z, θ configurations and beam sensor plane height h. The intersections are signified by filled circles, and the empty circles signify the intersection for $\theta = 0, Z = \Leftrightarrow 5$

Axis Parameterizations

As a consequence of the linearization assumption, we only need to compute x and y as linear functions of either the robot configuration z, or the beam sensor plane height h. The parameterizations $x = m_x z + b_x$ $y = m_y z + b_y$ involve four parameters m_x, b_x, m_y, b_y. The rotation axis is indicated by the superscript R (for example, m_x^R), the translations axis is indicated by the superscript T (for example, m_x^T), and the peg axis is indicated by the superscript P (for example, m_x^P).

Consequences of Linearization Assumption

The linearization assumption simplifies the analysis in three ways. First, it simplifies the task of finding the axis of rotation, because of the invariant that for various θ, the localized peg intersections form a circle (rather than an ellipse, since we made the linearization assumption (Figure 11.9)) centered at the intersection of the rotation axis and the beam sensor plane (Figure 11.7). Secondly, it simplifies the analysis of the axis of translation because we assume a linear relationship between the z height and the localized peg position. Consequently, it allows two-dimensional planar methods to solve the problem completely (Figure 11.8).

11.3 Parameter Estimation

In this section, we describe the technique for estimating each module's axes of rotation and translation. First we estimate each module's axis of rotation, and then we estimate each module's axis of translation. We are only able to determine the orientation of the translation axis, not its absolute offset, because the offset due to the axis and the offset due to the calibrating peg are indistinguishable; this does not pose a problem because we use a single calibration peg making the peg offset uniformly invariant.

11.3.1 Axis of Rotation

The axis of rotation is found by estimating the line connecting the centers of rotation at different sensor plane heights (Figure 11.10). Assuming that the peg was localized at diametrically opposite orientations, the intersection points average out to the center of rotation. The axis of rotation pose parameters are estimated by fitting lines to the x

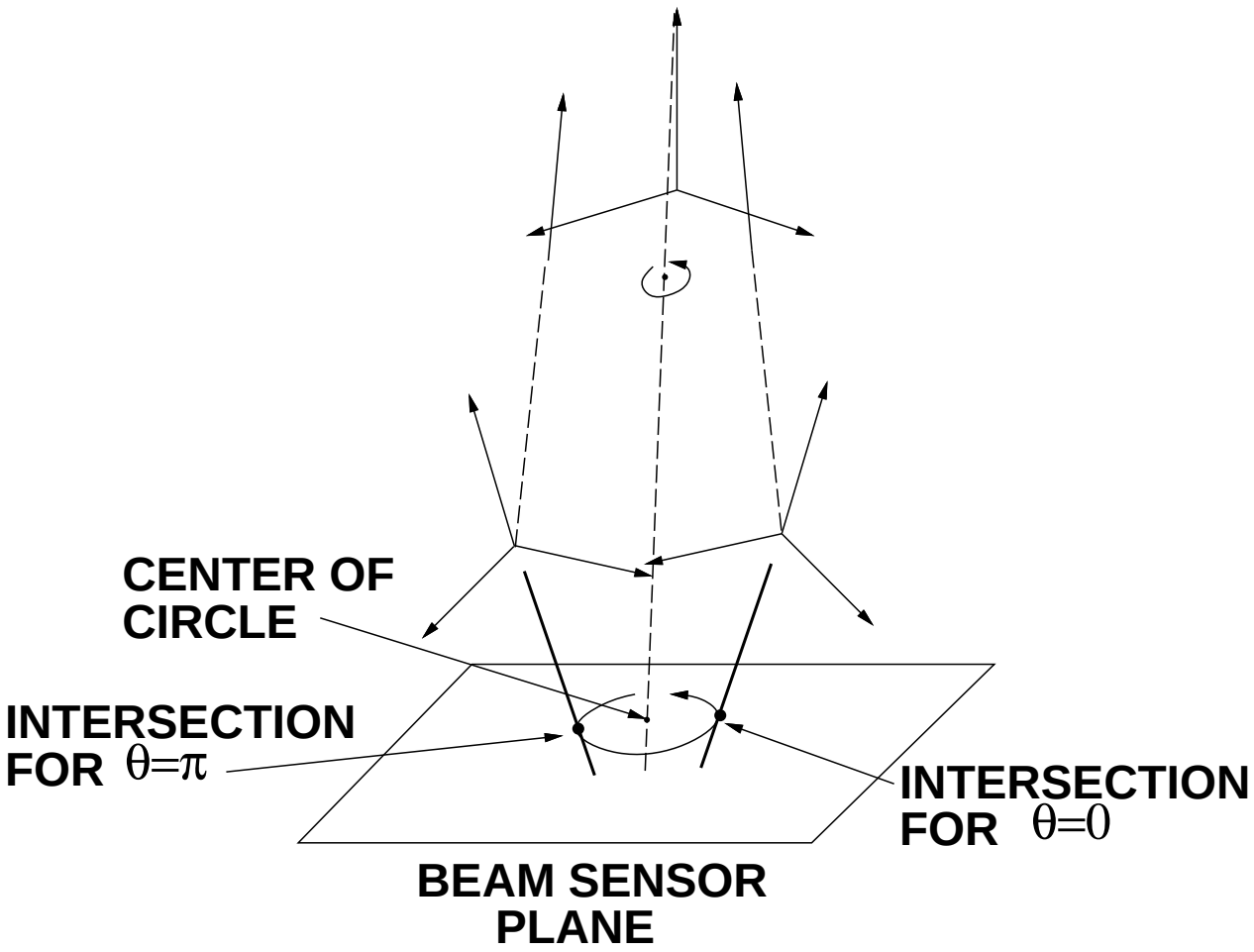


Figure 11.7: The intersection of the peg with the beam plane rotates as a function of θ around the intersection of the axis of rotation and the crossbeam sensor plane.

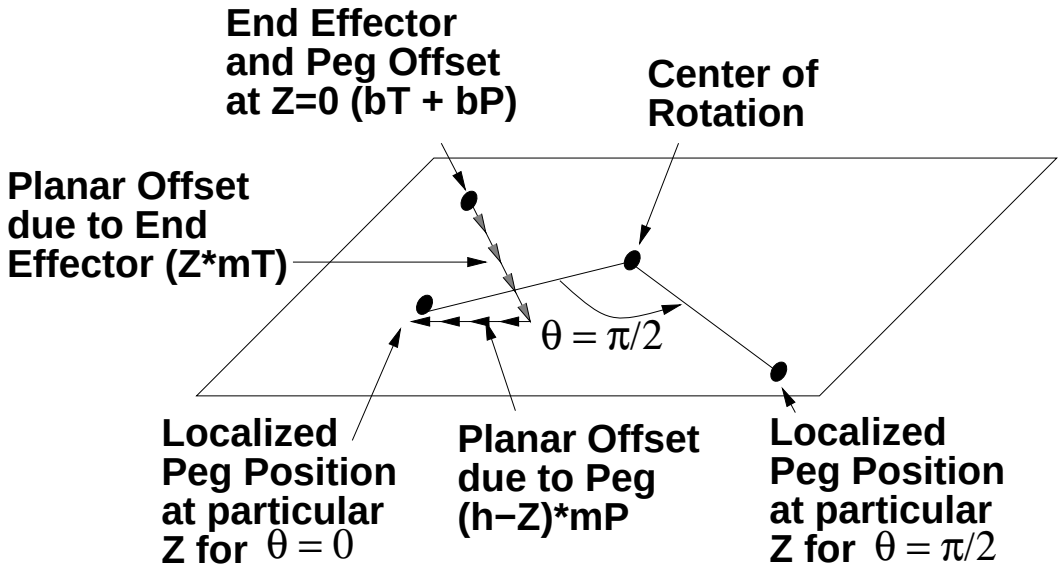


Figure 11.8: The problem can be solved using simple planar methods

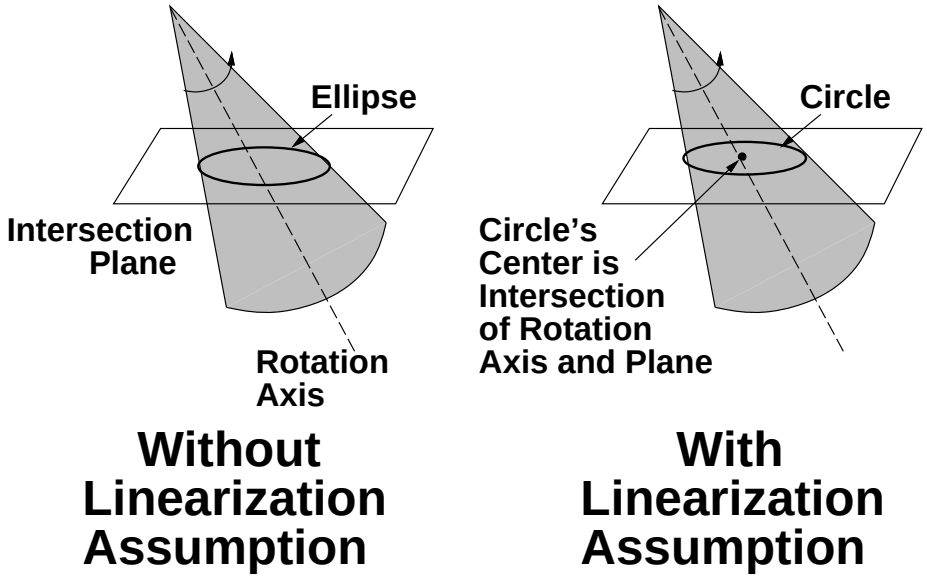


Figure 11.9: The linearization assumption implies that the intersection of a line rotated around a rotation axis and an intersection plane form a circle, rather than an ellipse irrespective of the tilt of the cone

coordinates and y coordinates of the localized peg position. (equations (11.1,11.2)).

$$I_x(z,\theta,h) = m_x^R h + b_x^R \quad (11.1)$$

$$I_y(z,\theta,h) = m_y^R h + b_y^R \quad (11.2)$$

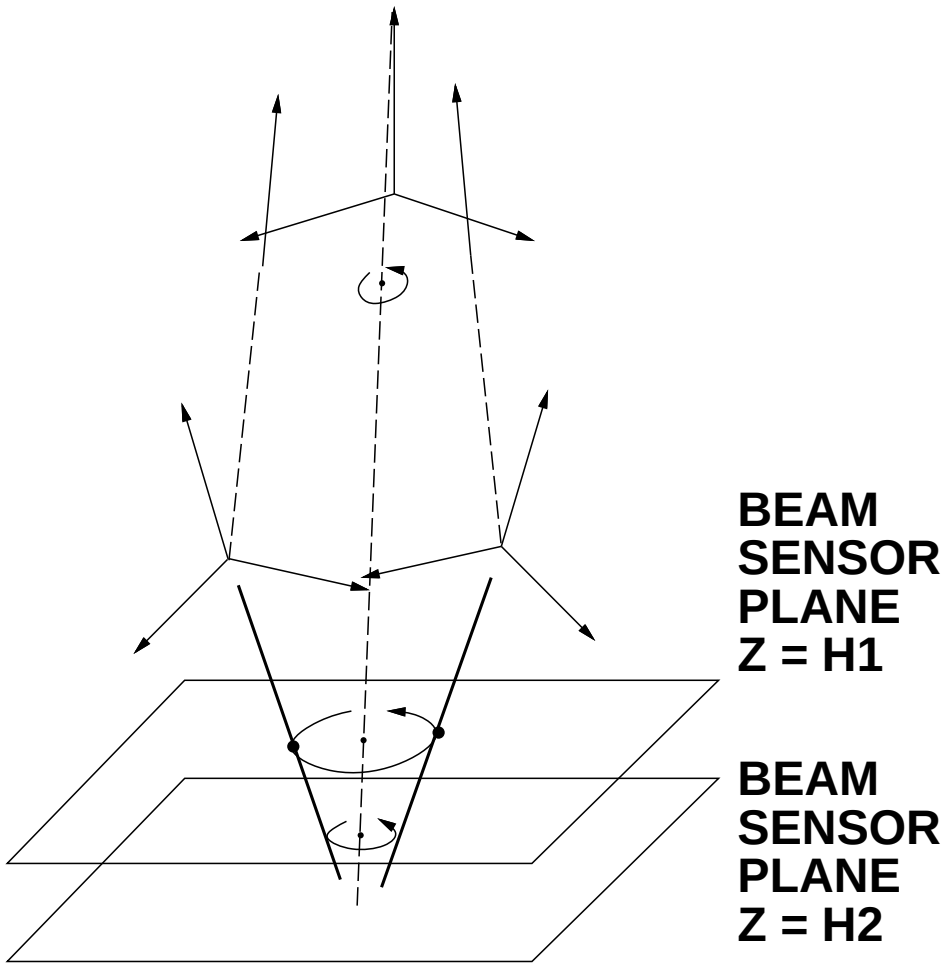


Figure 11.10: The axis of rotation is computed by fitting a line between the centers of rotations of the intersection points

11.3.2 Normalizing Data to $\theta = 0$

We use the axis of rotation estimate to remove θ dependence from the data set by normalizing all of the localized peg positions to configurations corresponding to $\theta = 0$.

This is accomplished by rotating each intersection (I_x, I_y) around the center of rotation (a function of z) by $\Leftrightarrow\theta$.

11.3.3 Axis of Translation

Determining the axis of translation involves using localized peg positions for various z configurations and beam sensor plane heights. Equation (11.3) predicts the intersection point as a function of the z height of the end effector and the height of the sensor plane h from the intersections (I_x, I_y) (Figure 11.11).

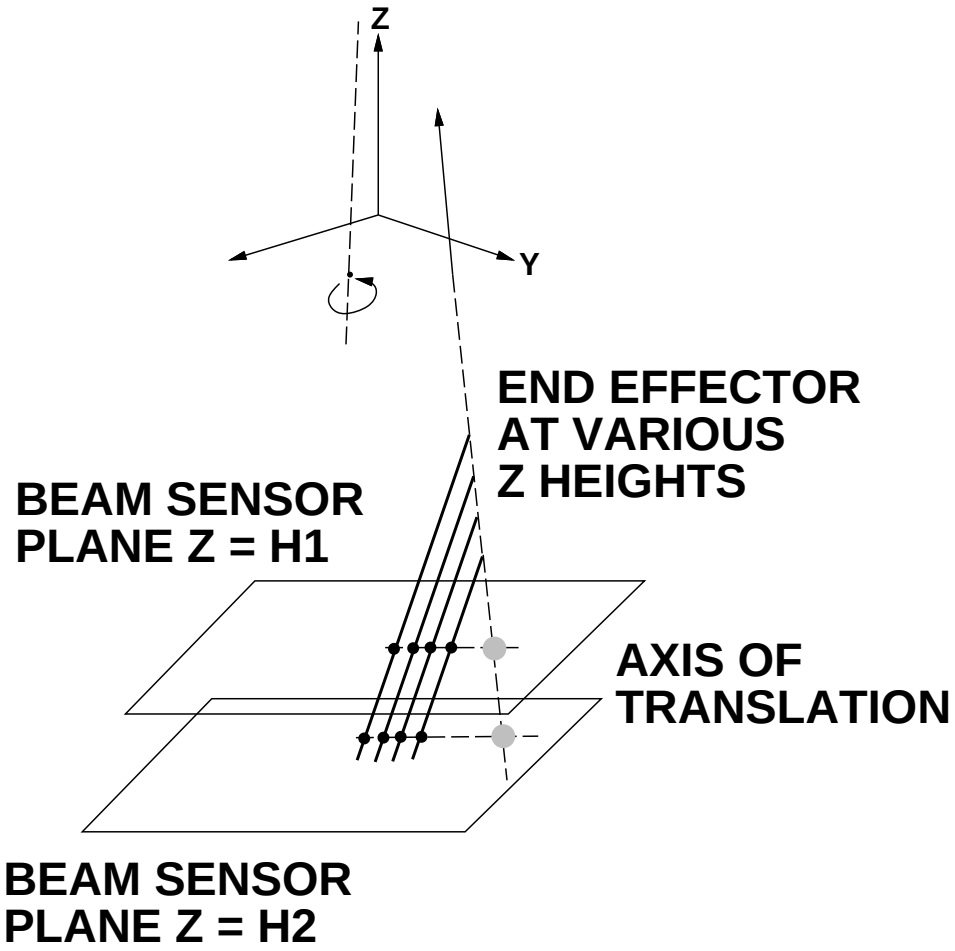


Figure 11.11: Method for estimating axis of translation

$$\begin{bmatrix} I_x(z,\theta=0,h) \\ I_y(z,\theta=0,h) \end{bmatrix} = \begin{bmatrix} m_x^T z + b_x^T \\ m_y^T z + b_y^T \end{bmatrix} + \begin{bmatrix} m_x^P (h-z) + b_x^P \\ m_y^P (h-z) + b_y^P \end{bmatrix} \tag{11.3}$$

The normalized intersection position is a linear function of z and h. Therefore, we can approximate the normalized data set by fitting planes to equations (11.4,11.5). The m_x^T,m_y^T are the sum of the z-dependent components and the h-dependent components. The translation offset b_x^T,b_y^T parameters are indistinguishable from the peg offset parameters, but this is allowable because the peg offset parameters b_x^P,b_y^P are constant for all calibration experiments.

$$I_x(z,\theta=0,h) = z(m_x^T \Leftrightarrow m_x^P) + b_x^T + b_x^P + h m_x^P \tag{11.4}$$

$$I_y(z,\theta=0,h) = z(m_y^T \Leftrightarrow m_y^P) + b_y^T + b_y^P + h m_y^P \tag{11.5}$$

11.3.4 Estimated Module Axis Parameters

For each module, calibration consisted of localizing the calibration peg for 960 different z,θ configurations using two crossbeam sensors at different heights. Table 11.1 presents the sum squared errors between simulated data predicted by the model, and the actual data, as well as the standard deviation (are values are in millimeters). Notice that module 0 and module 2 conform to the model quite well (discrepancies on the order of less than 0.025 mm), whereas module 1 does not conform as well. We assume that this implies that either the end effector arm is not straight, or that module 1 is loose due to wear. Still, this information is valuable, since determining wear is an important problem in its own right. Tables 11.2 and 11.3 shows the model parameter estimates.

11.4 Experiment and Results

To measure the robustness of our calibration technique, we performed cross-validation experiments, in which subsets of the peg position data were used to calibrate the modules, and then we measured how well the estimates predicted the entire data set.

Module	σ
0	0.01666
1	0.1048
2	0.0222

Table 11.1: Modeling errors for various modules

Module	m_x^R	m_y^R	b_x^R	b_y^R
0	0.000273	-0.000226	0.051043	-0.026717
1	0.005185	0.000844	0.657703	0.141354
2	0.002213	-0.001254	0.264434	-0.137513

Table 11.2: Rotation axis parameters for various modules

Module	m_x^T	m_y^T	$b_x^T + b_x^P$	$b_y^T + b_y^P$
0	-0.001128	-0.001287	-0.104147	0.004593
1	-0.000797	0.000269	0.025443	-0.081252
2	-0.001994	0.000379	-0.070300	-0.064175

Table 11.3: Translation axis parameters for various modules

11.4.1 Results

Figures 11.12 and 11.13 display the results of the cross validation experiments. The data suggests that as few as 40 experiments produce accurate calibration.

11.5 Conclusion

We focused on the calibration of a simple four degree of freedom (x, y, z, θ) manipulator, the Robotworld module because it combines high performance and simple calibration. Compared to general six degree of freedom manipulators, calibration of Robotworld modules was easier and more accurate. Calibration involved only modeling the Robotworld module's end effector arm's axis of rotation and axis of translation. These model parameters were estimated by passing an attached calibration peg through multiple crossbeam sensors at various z, θ configurations. This technique calibrated Robotworld modules to within 0.025 mm in terms of absolute position, as compared to 1 mm without kinematic modeling. We intend to experimentally verify calibration by inserting pegs into holes with 0.025 mm clearance via dead reckoning.

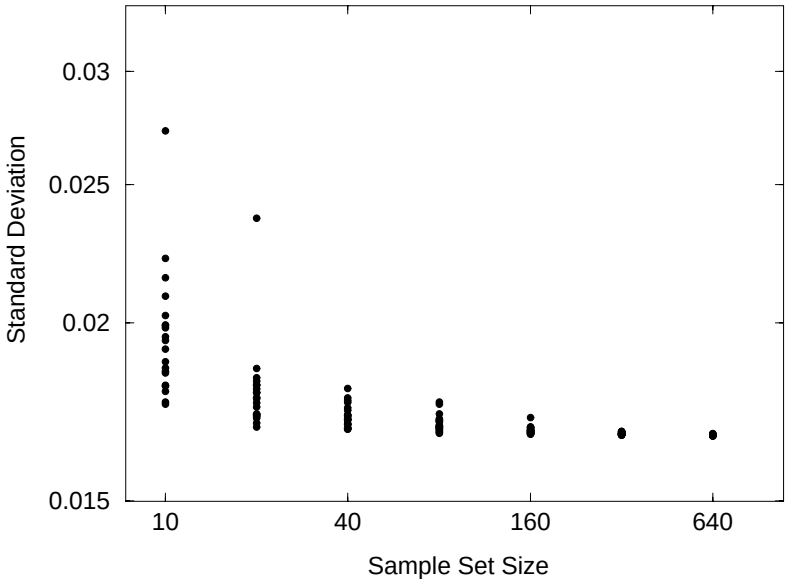


Figure 11.12: Standard deviation estimates for module 0 corresponding to calibration sets of various sizes measured via self validation

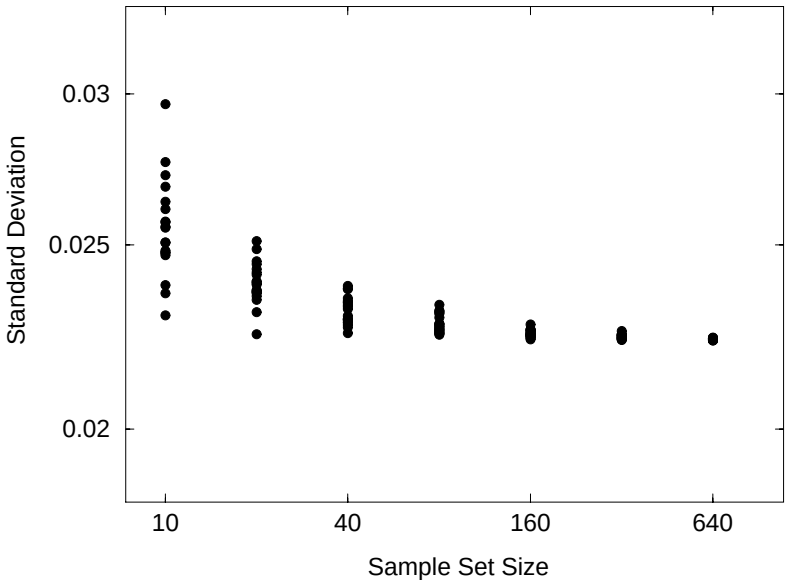


Figure 11.13: Standard deviation estimates for module 0 corresponding to calibration sets of various sizes measured via self validation

Chapter 12

Conclusion

The RISC (Reduced Intricacy Sensing and Control) Robotics philosophy, which aims to achieve high performance by forsaking generality, is the result of focusing on industrial and manufacturing applications (section 2.1.1). Stepping back, we see that RISC is a response to the customer (industry). We believe that manufacturing is mainly interested in speed, robustness, cost effectiveness, and precision, and less unconcerned about how these goals are achieved. Although generality and flexibility are very important, these issues are much more important to generic techniques than to manufacturing applications.

These limitations motivated us to investigate novel approaches to manufacturing tasks which relied on a combination of simple hardware and sophisticated algorithms. By rethinking, or reengineering the process in terms of each manufacturing task, we achieved high performance using elegant techniques.

In this thesis, we described how to address many tasks in manufacturing by a combination of simple hardware and sophisticated algorithms, rather than complex general-purpose hardware. Custom approaches have the advantage that the sensors/actuators can be manufactured to higher precisions, and provide greater preformance, robustness, cost effectivenes, and precision. Furthermore, high precision data/actuation greatly simplifies the algorithmic problems of interpreting the sensed data or enumerating valid designs.

Knowledge engineering systems and machine vision systems comprise a large fraction of manufacturing automation. Although these systems are attractive for a number of reasons, they have the drawback that they were originally developed to address the needs of fields other than manufacturing. Therefore, even specializations of these techniques may

not be able to overcome the limitations set by the original architect. Specilizations include automated fixture design expert systems and workcell integrated machine vision systems. For example, vision systems rely on camera image data, which can be coarse, inaccurate, and requires significant computational overhead. Additionally, automated fixture design assistants are usually expert systems, limited to executing rules in terms of machining features.

This research has focused on applying computer science techniques to two manufacturing tasks: machine vision, and fixturing. We approached both techniques by concentrating on the specifications, and their role in the entire process. We developed novel methods to perform these tasks using simpler devices and/or algorithms.

The main results of this research were generic indexing techniques used to identify objects, generic registration objects for localizing objects, and generic strategies for designing fixtures. These techniques all were all founded on computational algebra and computational geometry. We reduced each problem into a well understood related canonical problems. For example, the problem of constructing complete indexing tables is reduced to the problem of enumerating cells formed by an arrangement of curves [Ede87]. Furthermore, all of these strategies utilized a combination algebraic and numerical techniques to solve multivariate systems of equations [Man92].

12.1 Future Directions

In this thesis, we have proposed and solved many problems using a straightforward methodology: identify the degrees of freedom, identify the constraints, model the system, identify a finite set of critical configurations which can be determined analytically, and use a combination of algebraic, numerical, and geometric techniques to solve for these critical configurations. The advantage of this approach is that the problem is always manageable; *i.e.,* the problem's complexity is always bounded. This methodology is applicable to many problems in manufacturing, as well as many other domains.

This methodology, and the RISC Robotics philosophy leads in many directions: analytical sensor design, sensorless manipulation planning, parameterized object recognition, compact indexing table representations, etc.

The crossbeam sensing paradigm (Chapter 2) can be extended to handle a more expressive model class than generalized polyhedra. Extending the technique involves ex-

tending the generate and test formulations. Furthermore, more research should be done towards developing bette beam sensors; this would extend the applicability of crossbeam sensing to cover highly curved workpieces. Our commercially available beam sensors had beam aperture widths of a few millimeters. This posed a difficult problem because when curved objects were scanned, the individual beam sensors do not percieve the same cross-section.

The problem of analytical sensor design was suggested by Prof. Ken Goldberg and Prof. John Canny. Their goal was to determine the optimal beam orientations for a crossbeam sensor given a set of candidate models to be identified. I believe that this problem can be handled in the following manner: set an error threshold ϵ which corresponds to the minimum difference in any of the diameters/inset distance measurements necessary for disambiguation. Given that specification, we can check whether two object models are indistinguishable in polynomial time. Given ϵ, solve for the $b+1$ variables $\{\theta_{b_i} \Leftrightarrow \theta_{b_0}\}_{i>0}, \theta_{obj_j}, \theta_{obj_k}$ which satisfy the constraint that $b+1$ the discrepancies are equal to ϵ. Given all the crossbeam configurations $\{\theta_{b_i}\}, \theta_{obj_j}, \theta_{obj_k}$ for which the two models are indistinguishable, we can check the neighborhoods of each such configuration, and see if any neighboring configurations can disambiguate the two models. This can be done in polynomial time for each pair of models, and therefore can be done in polynomial time for all $\binom{n}{2}$ pairs of models by checking all of the neighborhoods around all of the critical points. Furthermore, we can find the optimal crossbeam sensor configuration, capable of maximally disambiguating the object models, by performing a binary search using ϵ.

I believe that sparse sensing techniques (Chapters 2 and 3), which may rely on a small number of measurements, can perform significant manufacturing tasks, such as accurately placing disk heads, extracting parts from a bag, etc. One of the difficulties of sparse sensing is that we cannot rely on feature based approaches, *i.e.,* matching sensed vertices to model vertices. In this thesis, we presented a generic indexing approach for solving the correspondence problem. The main advantage of the complete indexing approach is that it is straightforward, but that is also is greatest drawback. Often, there are symmetries or invariants which can be identified exploited so that a specialization achieves reasonable performance with less overhead; for example, Jacobs noticed that the two dimensional indexing manifold associated with each model feature group under affine transformation could be represented as the product of two one dimensional manifolds [Jac92].

Although I believe indexing tables are an important resource, there are other

important specialized correspondence algorithms waiting to be developed. For example, there are many types of objects (hexnuts, washers, screws) which come in a variety of sizes, and these objects can be considered families of parameterized models. Prof. Forsyth brought up the interesting idea of using alignment strategies for identifying parameterized model families from scanning sensor data. Since we can algebraically characterize the manifold of scanline intersections for a particular object, why not construct a family of manifolds for a family of parameterized models. Currently, it is not clear to me whether an indexing approach is feasible, although I can see how this can be done using alignment strategies.

The resultant-based localization strategies (Chapter 4) can be extended to handle three-dimensional localization from three-dimensional range data. Faugeras and Hebert [FH86] used exact methods to solve for the pose of each feature, and Sullivan *et al.* solved for the minimum total error pose using the iterative Levenberg-Marquardt algorithm [SSP94]. A resultant based approach which solved for the minimum total error pose would be more general and robust than either of these approaches Although Sullivan *et al.* handled generic algebraic curves, simple models of planes and quadrics probably suffice [FH86]. The resultant approach involves characterizing generic error functions between points and features of co-dimension one (planes, surfaces) as a function of the six degrees of freedom. After instantiating the generic error function, we then use a large resultant matrices to solve for the global minima. Of course, solving the associated eigenvalue problem will take a relatively long time because there are six degrees of freedom of degree at most six (squared error). Still, I am unclear on how to extend the approach to handle three-dimensional localization from two-dimensional perspective image data, because the error component associated with each data point corresponds to a different denominator. In order to use a resultant-based localization strategy, we need to approximate the error function by an algebraic expression.

Another important issue concerning resultants is the problem of degeneracies, or ill-conditioned matrix polynomials. Sometimes, due to the nature of the solutions of the multivariate system, the resultant matrix polynomial has an infinite solution space (a one parameter family of solutions). Currently, there are two approaches for overcoming this problem: generalized eigenvalue formulations, and performing Gaussian elimination on the matrix polynomial. Unfortunately, the generalized eigenvalue formulation does not provide sufficient numerical accuracy, and Gaussian elimination is performed on the matrix polynomial evaluated at a particular value with the caveat that the vectors in the null space of the matrix polynomial are also functions of the variable. Perhaps a better approach is to

perform singular value decomposition on the leading matrix $M^*(s)$ of the rational transform $t = \frac{as+b}{cs+d}$, and then multiply the other rational transform matrices by the SVD matrices. This approach should be immune to the problem associated with choosing a t value, because the leading matrix is still a function of t, although I have not yet tried this approach. Working with this problem firsthand, I think that a more robust approach is needed. Manocha has shown that c for generic cases, u-resultants constructed via the Macaulay formulation, produce matrix polynomials where the null-space should be independent of t, allowing one to extract an appropriate minor [Man94]. Another consideration is that it could be the case that there is another one parameter family of solutions to the system of partial derivatives, whereas the minimum is a single point; in this case, I believe that this system will produce an ill-conditioned resultant.

I am also interested in the problems of compacting indexing tables (Chapter 9). Our initial approach which utilized tree grids did not achieve satisfactory results with respect to both time and space. One unfinished and important element of our indexing algorithm is a method for storing similar lists in size $O(|A|)$ where $|A|$ is the number of elemetns, where the compaction can rely on sharing subtrees (memoization). This can be accomplished by a variant of the Sarnak and Tarjan algorithm; such an algorithm would further decrease the table storage space (and increase compaction) by a factor of $O(\log(m))$, the number of model feature groups.

Also of concern is the inability to efficiently trade off space and time for each other, to balance computational resources. I believe that there are many ways to improve the compaction performance of the tree grid technique, with the most advantageous involving taking all possible projection axes into account, and letting all of the model feature groups be classified via any of the projection axes. The hope is that by allowing each model feature group to be represented by any of its $b+1$ indexing coordinates, then coherence can be more markedly exploited, achieving higher compaction performance. I propose that the algorithmic problem of choosing the coordinate axis for each model feature group at each base point should be reduced in some way to a graph problem. Thereby, the weights in the graph would be associated with changes in the pairwise ordering between model feature groups associated with each coordinate axis. Thereby, the task becomes selecting a subset of graph nodes which minimize the total weight of the incident edges while consistently covering all of the nodes (model feature groups).

I am also interested in *complete* tree grid approaches. The current tree grid im-

plementation orders the model feature stratifications along a finite number of lines. This approach is incomplete because a stratification may not be completely characterizable by a finite set of points. Rather, we would like to characterize the stratifications by ranges of height intervals, providing a conservative estimate of the stratifications. For example, for the two dimensional base grid described in Chapter 9, we would intersect the stratification with a projection of each square in the base hyperplane, and the intersection would provide the range of height values consistent with each base square. These intervals could be ordered via an interval tree, without paying too high a cost in performance. The key observation is that the extreme values in each height interval are usually attributable to theprojections of the vertices of the square; there are only a linear number of extrema corresponding to the projections of non-vertex boundary points, and only a finite number of extrema corresponding to global extrema. By characterizing each of non-vertex extrema by a special token, we can still use generic tokens to characterize the interval tree endpoints, and therefore, share tree grid nodes.

Another problem related to work done by Prof. Ken Goldberg, deals with planning for sensorless manipulation. Prof. Goldberg developed a generic strategy for reorienting polygonal parts using a sequences of push grasp operations. This approach was generalized even further into an application-independent algorithm for sensorless manipulation for systems with a single degree of freedom. This approach relied upon a model wherein the operation mapped configurations within configuration space regions into a canonical point within the region. Prof. Goldberg and Dr. Rao proved that these types of systems admitted sensorless plans with linear numbers of steps in which each operation performing the mapping on a shifted copy of the configuration space. The basic idea is that each operation corresponds to mapping the cells an arrangement in configuration space to a point set, and we want to construct a sequence of such operations such that every point in configuration space will be mapped to a single point. I believe that this methodology can be generalized to higher dimensional configuration spaces. For each arrangement in configuration space, there are a finite number of translations which achieve a different mapping from canonical points to canonical points. Goldberg and Rao computed the operation sequence by working backwards from a canonical point and always increasing the size of the domain which mapped into that point, and I believe that the same approach is generalizable to higher dimensional configuration spaces.

In this thesis, I have presented many tools for localizing, calibrating, and fixtur-

ing, and I also think inexpensive high precision robots, such as Robotworld, afford sufficient accuracy to achieve generic assembly operations. Towards this goal, I have been involved in a number of piecemeal demonstrations which show how each step can be accomplished. Although I am proud of these demonstrations, I think that a more concrete demonstration, such as assembly of the model aeroplane engine or cassette player based upon general tools, would be an exciting and important achievement. I believe that robotic assembly demonstrations of this magnitude have become more and more feasible with recent results in robotics and manufacturing research. Paulos and Canny [PC94] showed 100% success at inserting pegs into holes with 0.025 mm tolerances; with such high precision, assembly planning can proceed in a dead reckoning fashion rather than sense and respond. I think the next step is to accomplish peg in hole insertions using two different robots using coordination, to provide a foundation for sensing and actuation, and also to validate calibration routines. After validating the coordination technique, the next step is to validate the performance of registration techniques, *i.e.,* machine vision, RISC sensors. The combination of precise coordination and and precise registration should result in accurate pick and place operations, which constitute a significant fraction of assembly.

I would also like to call attention to what I believe is the most important, least understood, area of industrial manufacturing: parts feeding. There are many hardware solutions (vibratory bowl feeders) for standard components (screws, nuts), but flexible manufacturing requires flexible parts feeding, and I think that this is a very important research area. Consider the impacts of parts feeding technology between computers and cellular phones, which mainly consist of electronic components and similarly sized electro-mechanical devices. I have noticed that for computers and cellular telephones prices decrease and quality improves much faster than for electro-mechanical devices such as VCRs and personal stereos. I believe that part of this difference is mainly due to the fact that electronic chips, the main component of computers and cellular phones, are fed in an automation friendly manner, spooled on tape, and therefore much easier to acquire than mechanical components, such as gears and levers.

Lastly, I want to list a few more areas in manufacturing where more research is necessary: process control, monitoring, user-friendly robot programming, failure detection methods, failure recovery methods, etc. All of these areas are critical to autonomous manufacturing.

Bibliography

[AB85]H. Asada and A. By. Kinematic analysis of workpart fixturing for flexible assembly with automatically reconfigurable fixtures. *IEEE Journal on Robotics and Automation*, RA-1(2):86–94, December 1985.

[ABB+92]E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0.* SIAM, Philadelphia, 1992.

[AF86]Nicholas Ayache and Oliver D. Faugeras. Hyper: A new approach for the recognition and positioning of two-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54, 1986.

[ATT67]"Description of a Robot Workspace Based on a Linear Stepper Motor". *AT&T Technical Journal*, 67(2):6–11, 1967.

[Bal81]D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13:111–122, 1981.

[BG94]Randy C. Brost and Ken Y. Goldberg. A complete algorithm for synthesizing modular fixtures for polygonal parts. In *International Conference on Robotics and Automation*, pages 535–542. IEEE, May 1994.

[BH91]D. J. Bennett and J. M. Hollerbach. Autonomous calibration of single-loop closed kinematic chanins formed by manipulators with passive endpoint constraints". *IEEE Transactions on Robotics and Automation*, 7:597–606, 1991.

[BM89a]Jin-Hwan Borm and Chia-Hsiang Menq. Determination of optimal measurement configurations for robot calibration based on observability measure. *International Journal of Robotics Research*, 10(1):51–63, 1989.

[BM89b]Jin-Hwan Borm and Chia-Hsiang Menq. Experimental study of observability of parameter errors in robot calibration. In *IEEE International Conference on Robotics and Automation*, pages 587–592, 1989.

[Bro88]R. C. Brost. Automatic grasp planning in the presence of uncertainty. *International Journal of Robotics Research*, 7(1):3–17, 1988.

[Bro91]R. C. Brost. *Analysis and Planning of Planar Manipulation Tasks*. PhD thesis, Carnegie Mellon University School of Computer Science, 1991.

[BWR93]J. Brian Burns, Richard S. Weiss, and Edward M. Riseman. View variation of point-set and line-segment features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):51–68, 1993.

[BYT90]John J. Bausch and Kamal Youcef-Toumi. Kinematic methods for automated fixture reconfiguration planning. In *IEEE International Conference on Robotics and Automation*, pages 1396–1401, 1990.

[CCB89]Y-C. Chou, V. Chandru, and M. M. Barash. A mathematical approach to automatic configuration of machining fixtures: Analysis and synthesis. *Journal of Engineering for Industry*, 111:299–306, 1989.

[CG94]John Canny and Ken Goldberg. RISC robotics. In *IEEE International Conference on Robotics and Automation*, 1994.

[Cha92]Chao-Hwa Chang. Computer-assisted fixture planning for machining processes. *Manufacturing Review*, 5(1):15–29, March 1992.

[CJ91]D. T. Clemens and D. W. Jacobs. Space and time bounds on indexing 3-d models from 2-d images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1007–1017, 1991.

[Col86]Richard Cole. Searching and storing similar lists. *Journal of Algorithms*, 7:202–220, 1986.

[Cra92]J. J. Craig. Robot calibration facilitates off-line programming. *Robotics World*, 10(1):24–25, 1992.

[CRC73]*Standard Mathematical Tables*. Chemical Rubber Company, 1973.

[CY87]Richard Cole and Chee K. Yap. Shape from probing. *Algorithmica*, 8:19–38, March 1987.

[DEY86]D. Dobkin, H. Edelsbrunner, and C. K. Yap. Probing convex polygons. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, pages 424–432, 1986.

[DF90]R. Deriche and O. Faugeras. Tracking line segments. *Image and Vision Computing*, 8:261–270, 1990.

[Dix08]A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[DP91]M. R. Driels and U. S. Pathre. Vision-based automatic theodolite for robot calibration. *IEEE Transactions on Robotics and Automation*, 7(3):351–36a0, 1991.

[Ede87]Herbert Edelsbrunner. *Algorithms in combinatorial geometry*. EATCS monographs on theoretical computer science. Springer-Verlag, 1987.

[EI93]Louis J. Everett and Thomas W. Ives. A sensor used for measurements in the calibration of production robots. In *IEEE International Conference on Robotics and Automation*, number 1, pages 174–179, 1993.

[EM86]M. A. Erdmann and M. Mason. An exploration of sensorless manipulation. In *IEEE Journal on Robotics and Automation*, pages 1569–1574, 1986.

[FB81]M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24:381–396, 1981.

[FC92]Carlo Ferrari and John Canny. Planning optimal grasps. *IEEE Journal on Robotics and Automation*, pages 2290–2295, 1992.

[FH86]O. D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-d objects. *International Journal of Robotics Research*, 5(3):27–52, 1986.

[FM90]O.D. Faugeras and S. Maybank. Motion from point matches: Multiplicity of solutions. *International Journal of Computer Vision*, 4:225–246, 1990.

[FMZ+91]D. Forsyth, L. Mundy, A. Zisserman, A. Heller, and C. Rothwell. Invariant descriptors for 3-d object recognition and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:971–991, Oct 1991.

[FP91]Bernard Faverjon and Jean Ponce. On computing two-finger force-closure grasps of curved 2d objects. In *IEEE International Conference on Robotics and Automation*, number 1, pages 424–429, 1991.

[GBDM77]B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[GCS91]Z. GIGUS, J. CANNY, and R. SEIDEL. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):542–551, 1991.

[GG73]M. Golubitsky and V. Guilleman. *Stable Mappings and Their Singularities*. Number 14 in Grad. Texts in Math. Springer Verlag, New York, 1973.

[GL89]G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins Press, Baltimore, 1989.

[GLP84]W. Eric L. Grimson and Tomas Lozano-Perez. Model-based recognition and localization from sparse range or tactile data. *International Journal of Robotics Research*, 3(3):3–35, 1984.

[GM90]Kenneth Y. Goldberg and Matthew T. Mason. Bayesian grasping. In *IEEE Journal on Robotics and Automation*, pages 1264–1269, 1990.

[GM91]Kenneth Y. Goldberg and Matthew T. Mason. Generating stochastic parts for a programmable parts feeder. In *IEEE Journal on Robotics and Automation*, pages 352–359, 1991.

[Gol92]Kenneth Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 1992.

[GQP93]Ambarish Goswanni, Arthur Quaid, and Michael Peshkin. Complete parameter identification of a robot from partial pose information. In *IEEE International Conference on Robotics and Automation*, pages 168–173, Atlanta, Georgia, 1993.

[Gri86]W. Eric L. Grimson. Sensing strategies for disambiguating among multiple objects in known poses. *International Journal of Robotics Research*, RA-2(4):196–213, 1986.

[Gri88]W. Eric L. Grimson. The combinatorics of object recognition in cluttered environments using constrained search. *International Conference of Computer Vision*, pages 218–227, 1988.

[Gri90]W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. MIT Press, Cambridge, MA, 1990.

[GT86]M. V. Gandhi and B. S. Thompson. Automated design of modular fixtures for flexible manufacturing systems. *Journal of Manufacturing Systems*, 5(4):243–252, 1986.

[GTG88]P. M. Grippo, B. S. Thompson, and M. V. Gandhi. A review of flexible fixturing systems in computer integrated manufacturing. *International Journal Computer Integrated Manufacturing*, 1:124–135, 1988.

[HK92]Mun Li Hong and Lindsay Kleeman. Analysis of ultrasonic differentiation of three dimensional corners, edges, and planes. In *IEEE International Conference on Robotics and Automation*, pages 580–584, 1992.

[HL93]John M. Hollerbach and David M. Lokhorst. Closed-loop kinematic calibration of the rsi 6-dof hand controller. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 142–148, 1993.

[Hor89]Bernard Klaus Paul Horn. *Robot Vision*. McGraw-Hill, seventh edition, 1989.

[Hor91]Bernard Klaus Paul Horn. Relative orientation revisited. *Journal of Optical Society of America*, 8(10):1630–1638, 1991.

[Hou62]P. V. C. Hough. Method and means for recognizing complex patterns, 1962. U. S. Patent 3069654.

[HU90]Daniel P. Huttenlocher and Shimon Ullman. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2):195–212, 1990.

[HW89]Caroline Hayes and Paul Wright. Automating process planning: Using feature interactions to guide search. *Journal of Manufacturing Systems*, 8(1):1–15, 1989.

[HW90]F. Brack Hazen and Paul Wright. Workholding automation: Innovations in analysis. *Manufacturing Review*, 3(4):224–237, December 1990.

[IS94]Jehuda Ish-Shalom. Sawyer sensor for planar motion systems. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2652–2658, 1994.

[ISW94]Jehuda Ish-Shalom and Tamer Wasfy. A finite element model for real-time compensation of the thermal deformation of the platen of a planar step motor. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1490–1495, 1994.

[Jac92]D. W. Jacobs. Space efficient 3d model indexing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 439–444, 1992.

[JJ90]C. Jerian and R. Jain. Polynomial methods for structure from motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1150–1166, 1990.

[KDN93]D. Koller, K. Daniilidis, and H.-H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, 1993.

[KG93]Dukhyun Kang and Kenneth Y. Goldberg. Grasp recognition strategies from empirical models. In *IEEE International Conference on Robotics and Automation*, number 3, pages 455–460, 1993.

[KSSS86]Alan Kalvin, Edith Schonberg, Jacob T. Schwartz, and Micha Sharir. Two-dimensional model-based boundary matching using footprints. *International Journal of Robotics Research*, 5(4):38–55, 1986.

[Lak78]K. Lakshminarayana. Mechanics of form closure. In *ASME Design Engineering Technical Conference*, 1978.

[LH90]Sukhan Lee and Hern S. Hahn. An optimal sensing strategy of a proximity sensor system for recognition and localization of polyhedral parts. In *IEEE International Conference on Robotics and Automation*, pages 1666–1671, 1990.

[LH91]Sukhan Lee and Hern S. Hahn. An optimal sensing strategy for recognition and localization of 3-d natural quadric objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1018–1037, 1991.

[LS88]Z. X. Li and S. Sastry. Task oriented optimal grasping by multifingered robot hands. *IEEE Journal on Robotics and Automation*, 4(1):32–44, 1988.

[LSW88]Yehezkel Lamdan, Jacob T. Schwartz, and Haim J. Wolfson. Object recognition by affine invariant matching. *International Conference of Computer Vision*, 1988.

[LW88]Yehezkel Lamdan and Haim J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. Technical Report No. 368, New York University, Robotics Research Laboratory, Department of Computer Science, May 1988.

[MA89]I. Mazon and R. Alami. Representation and propagation of positioning uncertainties through manipulation robot programs. integration into a task-level programming system. Technical Report 89042, LAAS (Laboratorie d'Automatique et d'Anlyse des Systemes, 1989.

[Mac02]F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

[Man92]Dinesh Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.

[Man94]D. Manocha. Computing selected solutions of polynomial equations. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 1–8, Oxford, England, 1994. ACM Press.

[Maz88]"Isabelle Mazon". Modelling positioning uncertainties. In G. Goos and J. Hartmanis, editors, *Geometry and Robotics Workshop: Lecture Notes in Computer Science no. 391*, pages 336–360. Springer Verlag, 1988.

[MC94]Brian Mirtich and John Canny. Easily computable optimum grasps in 2-d and 3-d. In *IEEE International Conference on Robotics and Automation*, San Diego, California, 1994.

[Mil85]"S. M. Miller". Impacts of robotics and flexible manufacturing technologies on manufacturing costs and employment. In Paul R. Kleindorfer, editor, *The Management of Productivity and Technology in Manufacturing*, pages 73–110. Plenum Press, 1985.

[Mis86]B. Mishra. Workholding: Anaylsis and planning. Technical Report No. 259, New York University, Courant Institute of Mathematical Sciences, November 1986.

[MMFF84]A. Markus, Z. Markusz, J. Farka, and J. Filemon. Fixture design using prolog: An expert system. *Robotics and Computer Integrated Manufacturing*, 1(2):167–172, 1984.

[MNP90]Xanthippi Markenscoff, Lugun Ni, and Christos H. Papadimitriou. The geometry of grasping. *International Journal of Robotics Research*, 9(1):61–74, 1990.

[MP89a]Xanthippi Markenscoff and Christos H. Papadimitriou. Optimum grip of a polygon. *International Journal of Robotics Research*, 8(2):17–29, 1989.

[MP89b]B. W. Mooring and S. S. Padavala. The effect of kinematic model complexity on manipulator accuracy. In *IEEE International Conference on Robotics and Automation*, pages 593–598, 1989.

[MSS87]B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

[NBWY93]Edward Nicolson, Steve Burgett, Aaron Wallack, and Barak Yekutiely. Optimal position sensing for closed loop control of linear stepper motors. In *JSME International Conference on Advanced Mechatronics*, pages 322–327, 1993.

[Ngu88]Van-Duc Nguyen. Constructing force-closure grasps. *International Journal of Robotics Research*, 7(3):3–16, 1988.

[Nic94]E. Nicolson. Standardizing i/o for mechatronic systems (sioms) using real-time unix device drivers. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 489–494, 1994.

[NO93]Wyatt S. Newman and David W. Osbron. A new method for kinematic parameter calibration via laser line tracking. In *IEEE International Conference on Robotics and Automation*, number 2, pages 160–165, 1993.

[NW78]James Nevins and Daniel Whitney. Computer-controlled assembly. *Scientific America*, 238(2):62–74, February 1978.

[Ohw87]E. N. Ohwovoriole. Kinematics and friction in grasping by robotic hands. *Transactions of the ASME*, 109, 1987.

[Ols94a]C. F. Olson. Probabilistic indexing: A new method of indexing 3d model data from 2d image data. In *Proceedings of the Second CAD-Based Vision Workshop*, pages 2–8, 1994.

[Ols94b]Clark Olson. *Fast Object Recognition by Selectively Examining Hypotheses.* PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1994.

[Ove92]M. Overmars. Point location in fat subdivisions. *Information Processing Letters*, 44:261–265, 1992.

[PC93]Eric Paulos and John Canny. Informed peg-in-hole insertion using optical sensors. In *SPIE Conference on Sensor Fusion VI*, 1993. Boston Massachusetts.

[PC94]Eric Paulos and John Canny. Accurate insertion strategies using simple optical sensors. In *IEEE International Conference on Robotics and Automation*, pages 1656–1662, May 1994.

[PHK92]Jean Ponce, Anthony Hoogs, and David J. Kriegman. On using cad models to compute the pose of curved 3d objects. *CVGIP: Image Understanding*, 55(2):184–197, 1992.

[PK92]John Ponce and David J. Kriegman. Elimination theory and computer vision: Recognition and positioning of curved 3d objects from range, intensity, or contours. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 123–146, 1992.

[PS85]F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[PVG93]J. P. Prenninger, M. Vincze, and H. Gander. Contactless position and orientation measurement of robot end-effectors. In *IEEE International Conference on Robotics and Automation*, number 1, pages 180–185, 1993.

[Reu76]F. Reuleaux. *The Kinematics of Machinery.* Macmillan, 1876.

[RG92]Anil S. Rao and Kenneth Y. Goldberg. Orienting generalized polygonal parts. In *IEEE International Conference on Robotics and Automation*, pages 2263–2268, 1992.

[RG93]Anil S. Rao and Kenneth Y. Goldberg. On the relation between friction and part shape in parallel-jaw grasping. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 461–466, 1993.

[Rob63]L. G. Roberts. *Machine Perception of Three-Dimensional Solids.* PhD thesis, Massachusetts Institute of Technology, 1963.

[Ros93]D. Rosen. Errors in digital area measurement of regular 2d figures. In *Vision Geometry II*, pages 26–32, September 1993.

[Sal85]G. Salmon. *Lessons Introductory to the Modern Higher Algebra.* G.E. Stechert & Co., New York, 1885.

[Sch79]Hilbert Schenck. *Theories of Engineering Experimentation.* McGraw-Hill Book Company, 1979.

[Sch87]Victor Scheinman. Robotworld: A multiple robot vision guided assembly system. In *Proceedings of the 4th International Symposium on Robotics Research*, 1987.

[Shi93]Bijan Shirinizadeh. Issues in the design of the reconfigurable fixture modules for robotic assembly. *Journal of Manufacturing Systems*, 12(1):1–14, 1993.

[SS87]Jacob T. Schwartz and Micha Sharir. Identification of partially obscured objects in three dimensions by matching noisy characteristic curves. *International Journal of Robotics Research*, 6(2):29–44, 1987.

[SSP94]S. Sullivan, L. Sandford, and J. Ponce. Using geometric distance fits for 3D object modelling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(12):1183–1196, December 1994.

[ST86]Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986.

[Sur90]Adwin H. Suryohadiprojo. Study of the error correction procedure in robot calibration. In *IEEE International Conference on Robotics and Automation*, pages 291–296, 1990.

[TM87]D. W. Thompson and J. L. Mundy. Three-dimensional model matching from an unconstrained viewpoint. *IEEE International Conference on Robotics and Automation*, 1:208–220, 1987.

[Wal95a]Aaron Wallack. Calibration of four degree of freedom robotworld modules. unpublished, 1995.

[Wal95b]Aaron Wallack. Constructing complete indexing tables to recognize polyhedral objects. unpublished, 1995.

[Wal95c]Aaron Wallack. Duality of calibrating point scanning and line probing sensors. unpublished, 1995.

[WC93]Aaron Wallack and John Canny. A geometric matching algorithm for beam scanning. In *SPIE Vision Geometry II*, pages 143–159, September 1993.

[WC94]Aaron Wallack and John Canny. Planning for modular and hybrid fixtures. In *International Conference on Robotics and Automation*, pages 520–527. IEEE, May 1994.

[WC95a]Aaron Wallack and John Canny. An indexing technique for sparse sensing strategies. IEEE International Symposium on Assembly on Task Planning, 1995.

[WC95b]Aaron Wallack and John Canny. Object recognition and localization from scanning beam sensors. In *IEEE International Conference on Robotics and Automation*, 1995. IEEE Conference on Robotics and Automation.

[WCM93a]Aaron Wallack, John Canny, and Dinesh Manocha. Object localization using cross-beam sensing. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 692–699, May 1993.

[WCM93b]Aaron Wallack, John Canny, and Dinesh Manocha. Object recognition and localization from scanning beam sensors. (preprint, University of California, Berkeley), 1993.

[WM94a]Aaron Wallack and Dinesh Manocha. Exact optimal pose determination using algebraic techniques. unpublished, 1994.

[WM94b]Aaron Wallack and Dinesh Manocha. Robust algorithms for object localization. Technical report, University of North Carolina, Chapel Hill, 1994.

[WN93]Aaron Wallack and Edward Nicolson. Optimal design of reflective sensors using probabilistic analysis. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 411–416, 1993.

[ZF90]Z. Zhang and O. Faugeras. Building a 3d world model with a mobile robot: 3d line segment representation and integration. In *International Conference on Pattern Recognition*, pages 38–42, 1990.

[ZGW94]Yan Zhuang, Ken Goldberg, and Yin Chung Wong. On the existence of modular fixtures. In *IEEE International Conference on Robotics and Automation*, pages 543–549, San Diego, California, 1994.