Darpa/Navy Grant No. N00014-94-1-0721

Dynamic Adaptation of Individual Perception-Action Control Plans in a Heterogeneous Team of Intelligent Mobile Agents

Final Report – June 1997

ORGANIZATION:

Stanford University

PRINCIPAL INVESTIGATOR:

Jean-Claude Latombe

Email: latombe@cs.stanford.edu

Tel: (415) 723 0350

Fax: (415) 725 1449

RESEARCHERS:

- Senior Research Associate: Barbara Hayes-Roth.

- Research Associate: Lydia Kavraki.

- Postdoctoral Students: Yoshihito Koga, Steven LaValle.

- Graduate Students: Craig Becker, Cyprien Godard, Hector Gonzalez-Banos, James Kuffner, David Lin, Karl Pfleger.

- System Administration: Frederic Cazals, Byung-Hyun Chung, Eugene Jhong, Junhe Lin, Ramin Penangwala, Prasanna Rawasami.

- Visitor: Rafael Murietta.

Executive Summary

The objective of this project was to explore the development of new type of physical agents, called **autonomous observers**. An autonomous observer is a mobile robot equipped with cameras that can perform vision tasks in response to high-level inputs given by human users. To perform such tasks, multiple observers may team to reach the same results quicker or to attain goals that no agent could achieve alone. For example, finding and/or tracking a fast target reliably in a complex environment may not be possible with a single observer.

Autonomous observers allow users to perform remote observation tasks (i.e., an important form of telepresence) without worrying about the details of camera motions. Instead, these motions are automatically computed and executed. The design and the implementation of the algorithms and architectural principles underlying autonomous observers yield new challenging problems in motion planning, control, and coordination, in which visibility conditions and motion obstructions must be simultaneously taken into account. Our research has addressed and solved some of these problems.

The need for autonomous observers arises in a variety of applications. For example, in the military domain, they can be used to assess and clear the situation in a building, to monitor and track motions of enemy targets, and to perform search/rescue operations in a potentially hostile environment. In medical surgery, surgeons often operate by watching graphic displays of key tissues; in that case, an autonomous observer could be used to maintain visibility of the tissues in spite of obstructions caused by people and complex mechanical instruments. Autonomous observers can also assist in distributed collaborations: researchers at one institution may want to conduct an experiment using robotic hardware at another institution; autonomous observers could then be used to gather and transmit crucial real-time information allowing the remote researchers to effectively monitor their experiment. Other applications include remote monitoring of manufacturing operations in an assembly plant, and supervision of automated construction efforts in space.

Our research has addressed three major topics: model building, target seeking, and target tracking. This sequence of three topics is implicitly based on the following hypothetical scenario: autonomous observers are dropped into an unknown environment, of which they first have to build a model (both for future navigation and for virtual fly-through on a graphic display); then they have to find a smart target hidding among view-obstructing obstacles; finally, they have to monitor this target and track its motions. However, the results that we have obtained on each topic can be separately used in other scenarios.

For each topic, we have developed and implemented new algorithms. We have experimented with our software both in simulation and with an autonomous observer prototype that we have designed and built.

Chapter 1

Overview

1.1 Scope and Motivation of Project

The original objective of the project was to develop broadly applicable technology for designing and implementing effective teams of heterogeneous intelligent mobile agents operating in dynamic environments. Initial cut in the project budget eventually led us to reduce the scope of our effort. Instead of conducting a broad, but superficial investigation of this topic, we focused our research on the development of a new type of physical agents, which we call **autonomous observers**. An autonomous observer is a mobile robot that is equipped with cameras in order to perform a variety of vision tasks such as building 2-D and/or 3-D models of unknown environments, finding unpredictable moving targets in environments cluttered by occluding obstacles, and tracking targets. An autonomous observer can use cameras for its own navigation (e.g., to localize itself or detect unexpected obstacles), but the main goal is to perform vision tasks specified by remote users. To perform such tasks, multiple autonomous observers may team up to achieve the same result quicker or to achieve goals that no observer could achieve alone. For example, finding and/or tracking a fast target reliably, in a complex environment, may not be possible with a single observer.

In other words, autonomous observers allow users to achieve remote observation (i.e., some form of telepresence) without worrying about camera motions. Instead, these motions are automatically computed and executed. The design and the implementation of the algorithms and architectural principles underlying autonomous observers yield new challenging problems in motion planning, control, and coordination, in which visibility conditions and motion obstructions must be simultaneously taken into account. Our research has addressed and solved some of these problems. But, perhaps even more interestingly, it has also uncovered a whole spectrum of new and broadly interesting problems for future research. We believe that

planning and controlling cooperative motions for achieving complex vision tasks will soon be regarded as a critical research area to achieve effective telepresence over a non-real-time computer network.

Indeed, the need for autonomous observers arises in a variety of applications. For example, in the military domain, they can be used to assess and clear the situation in a building, to monitor and track motions of enemy targets, and to perform search/rescue in a potentially hostile environment. In medical surgery, surgeons often operate by watching graphic displays of key tissues; in that case, an autonomous observer could be used to maintain visibility of the tissues in spite of obstructions caused by people and complex mechanical instruments. Autonomous observers can also assist in distributed collaborations: researchers at one institution may want to conduct an experiment using robotic hardware at another institution; autonomous observers could then be used to gather and transmit crucial real-time information allowing the remote researchers to effectively monitor their experiment. Other applications include remote monitoring of manufacturing operations in an assembly plant, and supervision of automated construction efforts in space.

These application tasks require performing several basic, high-level vision-oriented operations, such as locating and tracking a moving target, or automatic model construction. Although similar problems have been studied in other contexts, one distinguishing characteristic of our project is the need to satisfy geometric visibility constraints in the planning and execution of motion strategies. For example, prior work in visual tracking typically does not consider potential visual obstruction by obstacles [35, 60].

1.2 Technical Approach

Our research has addressed three major topics: model building, target seeking, and target tracking. This sequence of three topics is implicitly based on the following hypothetical scenario: autonomous observers are dropped into an unknown environment, of which they first have to build a model (both for future navigation and for virtual fly-through on a graphic display); then they have to find a smart target hidding among view-obstructing obstacles; finally, they have to monitor this target and track its motions. However, the results that we have obtained on each topic can be separately used in other scenarios.

In this section, we give a high-level perspective on our research, by outlining our technical approach of the three topics mentioned above. Note that implementing real autonomous observers requires addressing several other technical issues, such as reliable mobile robot navigation and image processing for tracking a pattern in an image sequence. Although we devoted a substantial amount of time to some of these issues, we think that they are less characteristic of our project's main objectives than our work on planning and controlling

motions under visibility constraints. Therefore, to keep our presentation focused, we will describe our work on peripheric issues in the appendices, and concentrate implementation details in Chapter 5: Implemementation of an Autonomous Observer Prototype.

1.2.1 Model Building

One of the most basic operations that autonomous observers may have to perform is to build a representation of an environment using vision sensing. If the only purpose of this model was to facilitate subsequent navigation of the observers, a two-dimensional model would be sufficient in most cases. But a useful application of the model is to allow remote users to "fly-trough" a virtual representation of the environment. Hence, our main goal has been to devise techniques to construct a model combining 3-D geometry with texture maps to produce realistic graphical renderings for virtual fly-through operations. This model, or part of it, can also be used for target finding and target tracking. Each sensing operation, which requires acquiring 3-D and texture data and merging this data with a current partial model, is rather expensive, so that creating an entire environment model can be quite time consumming. Therefore, part of our research has addressed strategic planning issues aimed at reducing the number of sensing operations.

A classical problem in automatic model building using a vision sensor is known as the *next-best-view problem*: Where to place the sensor next to maximize the amount of information that will be added to the partial model built so far? This problem has attracted considerable attention (e.g., see [3, 17, 54, 62, 76]), but techniques to solve this problem are not ideal for autonomous observers. One reason is inherent to the problem itself: the next-best-view problem is a *local* planning problem [39], so that a sequence of next-best views to build a complete model would often yield a prohibitive number of sensing operations. Moreover, a limitation of current next-best-view techniques is that they assume precise localization of the sensor. Merging two partial models first requires 3D data from these models to be aligned through partial matching. With mobile-robot observers, odometric or GPS errors between successive views may cause significant misalignments resulting in incorrect models. Fixed scanners used to generate models for small objects (e.g., as in [73]) do not raise this difficulty, because their positioning is far more precise. Finally, due to physical obstructions, a next-best-view technique may not suggest viewing positions that are accessible to the observers.

These remarks have led us to devise a new approach to model building. Assume for a moment that we are given a 2-D map of the environment describing the geometry of a horizontal cross-section at approximately the height of an observer's camera. A classical art-gallery algorithm [58] then computes a small number of positions such that if an observer successively visits each one of these positions, it eventually sees the entire 2-D environment (assuming

omnidirectional and illimited-range vision). The idea of our approach is simply to send autonomous observers to these locations, and to collect and merge data. If a single observer is available, it visits the locations in some sequence; if multiple observers are available, they can simultaneously collect data at several location.

Of course, the principle of this model-building approach requires several refinements:

(1) The fact that the entire 2-D environment is visible from a set of positions does not in general entail that the entire 3-D environment is also visible. For most indoor environments, however, this is almost true. A number of "holes" will usually remain in the model built, but these are usually small enough to be filled by adding a few sensing operations at locations computed using a next-best-view technique.[1]

(2) Art-gallery algorithms strive to minimize the number of positions where to place the observers. However, 3-D/texture sensing at these positions can yield partial models that have very small overlap between them. As mentioned above, it is suitable to align 3-D data from two partial models before merging them. Reliable alignment requires some minimal overlap between the two models, which requires some adaptation of typical art-gallery algorithms.

(3) Most art-gallery algorithms use a simple "line-of-sight" visibility model: one point sees another if the line segment between them does not cross any object. However, imperfections in vision sensors require that we use a more realistic definition taking distance and incidence into account. Like in remark (2), this yields new variants of the art-gallery problems.

(4) The 2-D model must be built in the first place. We do this by letting the observer navigate in the environment, using a simple laser range sensor projecting a horizontal plane of light to obtain the environment's 2-D contour. Since this form of sensing is fast, the number of sensing operations is not critical. Since furthermore we don't have prior knowledge at this stage a next-best-view technique to decide on-line the successive viewing positions is then appropriate.

Of course, one can imagine variants of this overall approach. A useful variant would be to interweave the construction of a 2-D model and that of a 3-D model to minimize the distance travelled by a robot. Another useful variant would be to construct the minimal model that contains objects specified as inputs. We have not investigated such variants, but we believe that the techniques we have developed could be used, in combination with others, to solve them.

Chapter 2 presents our results in model building in detail. It describes a set of complementary techniques (which we have implemented into an integrated software module) for merging several 3-D images into a single 3-D mesh model and for mapping texture patches extracted from color images onto the elements (triangles) of this mesh. To illustrate, Figure 1.1 shows

[1]Since a mobile observer is usually not a free-flying device, some holes can never be eliminated.

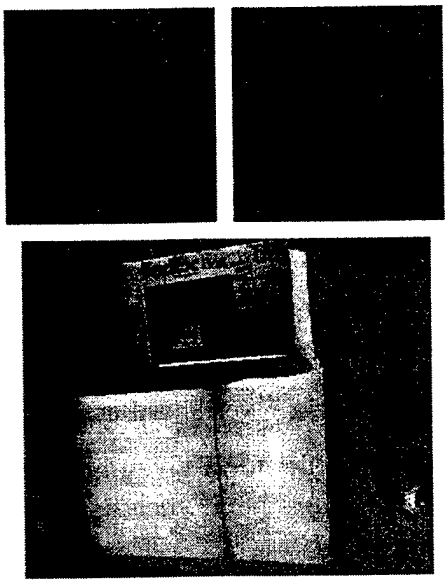


Figure 1.1: A preliminary result in map building

a model built by this module. This model was constructed from two different views: the scene is a pile of boxes with a coffee mug to the right. The 3-D data in the two partial models have been aligned, then fused together, prior to texture mapping. Chapter 2 will also introduce some of the issues about planning strategies for the autonomous observers where they should acquire images of the environment. A subset of the corresponding algorithms have been implemented and tested so far. Chapter 2 is complemented by Appendix B.

1.2.2 Target Finding

Let us now suppose that a model for the environment is available. We would like our observers to visually locate a potentially-moving target. In some applications, a strong predictive model might be given for the target, which can greatly simplify the target-finding task. However, in many applications no such models, or only weak ones, are available (e.g., in military situation assessment and surveillance application). Therefore, we assume that the target is unpredictable, has unknown initial position, and is capable of moving arbitrarily fast.

The target-finding problem is to plan a motion strategy for one or more observers in a given environment to eventually see a target. A visibility model defines the visibility region of the observers at any of their possible positions. Ideally, we are seeking a motion strategy

that guarantees that the target will ultimately lie in this visibility region. As the observers move, the visibility region deforms according to the observer's position and obstructions. The motion strategy must be such that, as the observers move, their visibility region sweeps the environment so that the target has no remaining place where to hide. Although related problems have been studied in other contexts [61, 68], this represents a novel robot planning problem.

Important questions are raised by this problem:

(1) What is the minimum number of observers needed for the existence of a guaranteed motion strategy? (Clearly, a guaranteed strategy exists if the number of observers is illimited.)

(2) If a guaranteed motion strategy for a given number of observers exist, how to plan it?

(3) If no guaranteed motion strategy exist for a given number of observers, how to generate a strategy that achieves a weaker requirement, such as minimizing the space in which the target can still hide.

The first question is important because of the cost of each observer and the complexity of coordinating multiple observers. Note that the answer depends only on the geometry of the environment. In our investigation so far, we have considered the case of 2-D polygonal environments of arbitrary complexity with and without holes, and the simple line-of-sight visibility model. We have established lower bounds which state that the number of needed observers can be logarithmic in the number of environment edges (geometric complexity) for a simply-connected free space and can be the square root of the number of holes for a multiply-connected free space (topological complexity). We have also established upper bounds which state that the number of observers is at most logarithmic in the number of environment edges for a simply-connected free space, and is at most linear in the number of holes for a multiply-connected free space.

To address the second question above, we have designed and implemented a complete planning algorithm, again for 2-D polygonal environments and line-of-sight visibility model. This algorithm precomputes a cell arrangement by identifying line segments that cause critical changes (topological variations) in the visibility region of the observers. A solution computed by this planner is shown in three frames in Figure 1.2 for one observer. The thick curve shows a portion of the observer's trajectory. Gray regions are the visibility regions of the observer at the positions attained in each frame. Black regions represent places where the target might still be hiding, and white regions represent the cleared area. The thin lines indicate cell boundaries.

Up to now, all our experiments have been done in simulation. The algorithm is quite efficient for problems that can be solved by a single observer. If the problem requires multiple observers, we currently use a greedy technique for combining the searches of individual

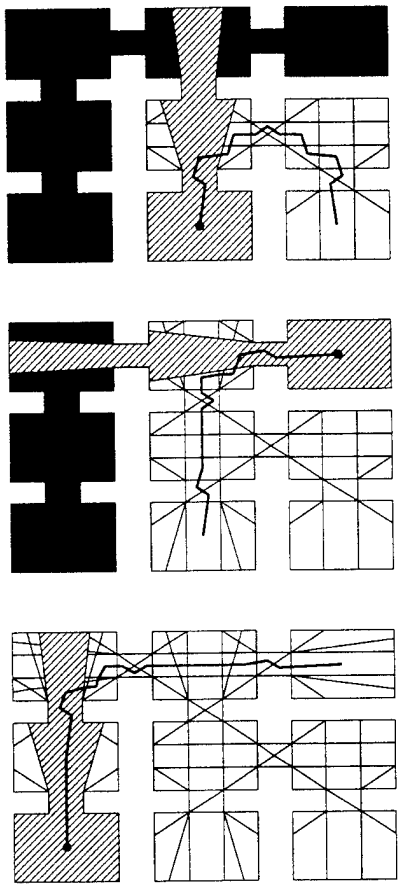


Figure 1.2: A computed target-finding strategy

observers. This technique works as follows: A first observer tries to clear the largest possible portion of the workspace (measured by the number of cells in which the target can no longer be). Then a second observer comes into action and does the same for the remaining space. And so on. However, the multi-observer algorithm is somewhat more sophisticated. Indeed, after a new observer is introduced, it may become possible for a previous observer to resume moving and increase the size of the region it can clear.

Our algorithms solve many hard problems efficiently. But when multiple observers are needed and our greedy algorithm finds a solution with more than two observers, we cannot guarantee that the number of observers is minimal. Applying the greedy algorithm with a given number of observers may yield a non-guaranteed strategy. But, in that case, the strategy minimizes the size (in number of cells) of the region where the target still hides.

We are currently studying several variations. In one, we incorporate a more realistic visibility model (i.e., limited field of view). In another, we try extend our algorithms to 3-D workspace.

1.2.3 Target Tracking

Once a target has been found, the next logical step is to maintain visibility with the target by appropriately moving the observers, again taking visibility and motion obstructions into account. Unlike with target finding, time is critical. The faster the planner and the more efficient the motion strategies, the better. But these two goals – fast planner vs. efficient strategies – are conflicting. This led us to develop and experiment with several planning algorithms to adapt to different tracking conditions; these algorithms, however, are based on similar principles. Our algorithms also try to optimize a given criterion such as the total distance traveled, energy utilized by the observer, or the quality of the visual information.

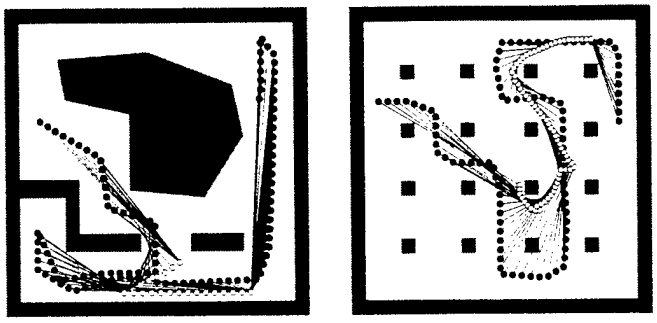
Our algorithms can be divided into two categories, on the basis of whether the target is predictable, or not:

1. For the predictable case, our algorithm runs off-line and computes an optimal solution (for the given criterion).
2. When the target is only partially predictable (e.g., we might only know its maximum speed or acceleration), two on-line variants have been developed that each attempt to maintain future visibility with limited prediction:
   - One computes a strategy by maximizing the probability that the target will remain in view in a subsequent time step.
   - The other maximizes the minimum time in which the target could escape the visibility region.

We have also considered the intermediate case where the target is restricted to move along a predefined network of paths, but its decision at each node of this network is unpredictable.

Unlike for target seeking, these algorithms can easily adapt to different and realistic visibility models. In most of our work, we have assumed that visibility is limited to a cone of given angle and bounded by some maximal distance.

We have implemented these algorithms and performed numerous successful experiments both in simulation and with our autonomous observer system. Figure 1.3 shows two simulation examples computed by the off-line planner. The target is displayed as a black disc and



a. b.

Figure 1.3: Optimal target tracking strategies.

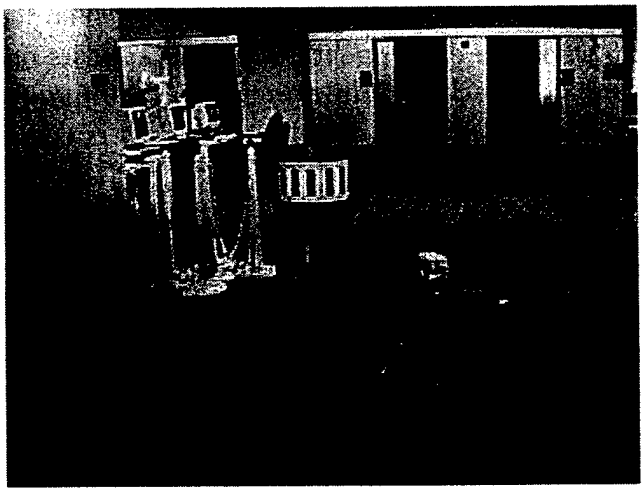


Figure 1.4: Our mobile robots used in the experiments.

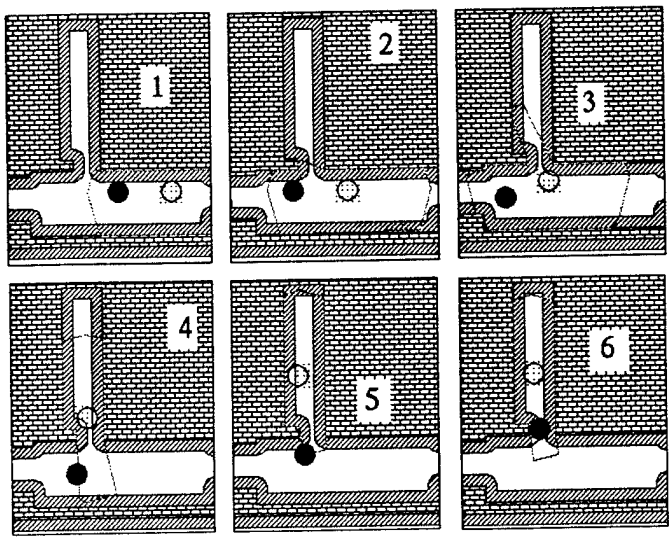


Figure 1.5: The on-line planner interface during experimentation

the observer as a white disc. Gray areas indicate obstacles creating visibility and motion obstruction. In Figure 1.3(a), the planner generates a tracking trajectory that minimizes the total distance traveled, while in (b) it minimizes the time during which the observer does not see the target under the additional constraint that the observer's speed is only half that of the target. Figure 1.4 shows our experimental setup: the robot in the forefront is the observer, while the other robot, with a distinctive "hat" used to facilitate target tracking, is the target. Figure 1.5 shows an experimental run as it appears on the user's display (the gray disc is the target and the black disc the observer). Visibility computations are done in a 2-D model of the workspace.

Our algorithms apply to one or several observers. However, our optimal off-line planning algorithms take exponential time in the number of observers and take prohibitive time for more than two observers. On the other hand, on-line algorithms are basically linear in the number of observers.

In Chapter 4, we will describe our target-tracking planning algorithms in detail. We will also present experimental results, both in simulation and with real robots. We have also developed software to track more complex targets (e.g., humans); see Appendix C.

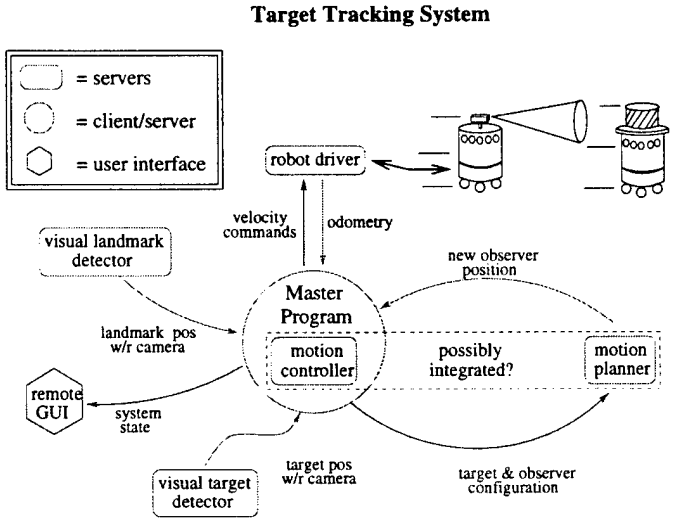


Figure 1.6: Architecture of the Target-Tracking System

1.3 Experimental Systems and Software Modules

We have built three separate experimental systems, for model building, target finding, and target tracking. The target-finding system works in simulation only. The other two operate in real 3-D environments.

The architectures of these systems and the software modules they contain will be described in more detail in the following chapters. The target-tracking system illustrates well the architecture of a complete autonomous observer. This architecture is shown in Figure 1.6. It combines of a rather large number of software modules running on different processors. At a high architectural level, these modules include: a landmark-based navigation module (described in Appendix A), a visual target tracking module, the target-tracking motion planner, and a motion control module. The navigation, visual tracking, and motion control modules run on on-board processors mounted on the autonomous observer. The planner runs on a fixed workstation connected to the network. The observer is equipped with a network radio interface. The overall system is accessible through a Web interface. Chapter 5 describes the implementation of an autonomous observer prototype.

Our three systems consist of many software modules, which could be of independent interest, e.g.: robust visibility analysis in 2-D workspace; landmark detection, recognition, and localization; visual tracking of a simple geometric pattern in an image sequence; visual tracking of a natural 3-D pattern in an image sequence; acquisition of 3-D models using a camera-laser sensor; fusion of overlapping 3-D models (triangular meshes).

1.4 Main Contributors

The main contributor for Model Building is Héctor González-Baños.

Target Finding is the work of Steve LaValle and David Lin.

The contributors for Target Tracking are Craig Becker, Steve LaValle, David Lin, and H. González-Baños.

Implementation of the autonomous observer prototype was achieved by Craig Becker, H. González-Baños, and David Lin. The motion planner, landmark detector and the target tracker was coded by Craig Becker. The graphical user interface to the system and operation thru the web is the work of David Lin. Asynchronous control of the robot and integration of system-state information was done by H. González-Baños.

The main contributor to the appendix on Landmark-Based Navigation is Craig Becker.

The appendix on Range-Image Acquisition is the work of Héctor González-Baños.

The main contributor to the appendix on Pattern-Tracking in an Image Sequence is Rafael Murrieta.

Prof. Leo Guibas, Rajeev Motwani, and Carlo Tomasi have also contributed to several aspects of this project.

1.5 Exchanges and Technology Transfers

1.5.1 Interaction with Other Grants

- The landmark-based navigation techniques were derived from previous work done under grant N00014-92-J-1809.

- The notion of cooperating autonomous observers have led us to identify visibility-based motion planning problems as being of great interest for military applications. We currently investigate several such problems under ARO MURI grant DAAH04-96-1-007 and a new NSF grant.

- We believe that one important application of autonomous observers is to help geographically distributed researchers to perform joint experiments in robotics. In this context observers are used to acquire pertinent data about the ongoing experiments and to show relevant images to remote researchers. In February '97 we have submitted a new proposal to NSF on this application, in collaboration with Prof. Bajcsy (University of Pennsylvania).

1.5.2 Publications, Seminars, and Web Sites:

• We have published or submitted the following invited or refereed articles:

C. Becker, J. Salas, K. Tokusei, and J.C. Latombe. Reliable Navigation Using Landmarks. *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 401-406.

C. Becker, H.H. González-Baños, J.C. Latombe, and C. Tomasi. An Intelligent Observer. *Lecture Notes in Control and Information Sciences*, Proc. 4th Int. Symp. on Experimental Robotics, Stanford, 153-160, June-July 1995.

J.C. Latombe. Controllability, Recognizability, and Complexity Issues in Robot Motion Planning. *Proc. 36th Annual Symp. on Foundations of Computer Science (FOCS)*, 484-500, October 1995.

S.M. LaValle, H.H. González-Baños, C. Becker, and J.C. Latombe. Motion Strategies for Maintaining Visibility of a Moving Target. *IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997.

S.M. LaValle, D. Lin, J.C. Latombe, L.J. Guibas, and R. Motwani. Finding an Unpredictable Target in a Workspace with Obstacles. *IEEE Int. Conf. on Robotics and Automation*, Albuquerque, NM, 1997.

L.J. Guibas, J.C. Latombe, S.M. Lavalle, D. Lin, and R. Motwani. A Visibility-Based Pursuit-Evasion Problem. Invited paper in the Special Issue on the CGC Workshop on Computational Geometry, *International Journal of Computational Geometry and Applications*, 1997.

L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, and R. Motwani. Visibility-Based Pursuit-Evasion in a Polygonal Environment. *Proc. Workshop on Algorithms and Data Structures (WADS'97)*, 1997.

H.H. González-Baños, J.C. Latombe, S.M. LaValle and D. Lin. Motion Planning with Visibility Constraints: Building an Autonomous Observer. Accepted for publication at the *8th Int. Symp. on Robotics Research*, Osaka, Japan, October 1997.

K. Pfleger and B. Hayes-Roth. Plans Should Abstractly Describe Intended Behavior. In A. Meystel, J. Albus, and R. Quintero (eds.), *Intelligent Systems: A Semiotic Perspective*, Proc. Int. Multidisciplinary Conf., Vol. I: Theoretical Semiotics, 29-34, NIST, Gaithersburg, MD, 1996.

B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic. A Domain-Specific Software Architecture for Adaptive Intelligent Systems. *IEEE Trans. on Software Engineering*, 21(4):288-301, 1995.

• We also have given the following additional seminars and presentations:

S.M. LaValle. A Visibility-Based Pursuit-Evasion Problem. CGC Workshop on Computational Geometry. John Hopkins U., October 1996.

S.M. LaValle. Visually Locating and Monitoring Moving Targets in Cluttered Environments. U. of Pennsylvania, October 10, 1996; CMU. October 15, 1996; U. of Illinois at Urbana-Champaign, October 18; U.C. Berkeley, April 1, 1997.

S.M. LaValle. Game-Theoretic Motion Planning with Emphasis on Algorithms for Visibility-Based Tasks. Iowa State University, March 20, 1997.

• Our results are presented in the following (linked) web pages, which include Postcript files of several papers as well as Java applets:

Papers only: http://robotics.stanford.edu/ latombe/pub.html#G

Short description, papers, and applet: http://robotics.Stanford.EDU/ latombe/projects/#D

Long description: http://robotics.stanford.edu/groups/mobots/home.html

Status page: http://robotics.stanford.edu/users/io/

Architecture of Autonomous Observer and description of software modules: http://robotics.stanford.edu/users/io/io_architecture.html

1.5.3 Joint Experiments with Other Laboratories

We have integrated a complete prototype of an autonomous observer with target tracking capabilities. The functions of this system are accessible through an interface on the Web. Using this system we have conducted experiments with Prof. Bajcsy's group (as part of a small project called ANVIL jointly supported by NSF and DARPA) at the University of Pennsylvania, with Prof. J.L. Gordillo at ITESM, Monterrey, Mexico, as part of a two-year project jointly funded by NSF and CONACyT, and with Prof. S. Hutchinson's group at the University of Illinois (Urbana-Champaign). The prototype system used in these experiments is the one described in Chapter 5.

1.5.4 Technology Transfer

We have transferred our landmark-based navigation technology to Nomadic Technologies that now markets it with its robotics products. Nomadic Technologies is a company based in Mountain View (CA) which designs and markets mobile-robot products. This transfer

consisted on training Nomadic Technology engineers on the new technology, improving the software, and cooperatively building an experimental demonstration. The work was done under a separate STTR DARPA grant supporting Nomadic Technologies and Stanford University (J.C. Latombe and C. Tomasi).

Our general research in motion planning (for which N00014-94-1-0721 provides partial support) has led multiple transfers to other companies. We have transferred path planning technology to General Electric Research (Schenectady, NY), where it is used to check that designated parts can be removed from an aircraft engine for inspection and repair. GE has reported a gain of 500% in planning efficiency. Our planning techniques have been integrated by GE engineers in a larger software system that GE has made available to other companies like Boeing. We have transferred a similar planner to General Motors Research Labs (Warren, MI) for testing part removability in automotive assemblies. In another collaboration with Pfizer Pharmaceuticals, we have adapted motion planning technology to design and implement tools to help chemists select promising drug molecules; in this work, molecules are treated as kinematic structures under the influence of force fields.

1.5.5 Military Interactions

Jean-Claude Latombe visited ARL, Aberdeen Proving Ground (MD) and gave a presentation in March '96.

CWO L.R. Cook (Fort Belvoir, VA), Major C. Hunt (Fort Belvoir, VA), and Major J.T. Girard visited our group in February '97. We gave them a presentation of our work on autonomous observers.

Jean-Claude Latombe participated in the DARPA SMART meeting (Dulles Airport) in May 1997.

1.5.6 Other Interactions

We have interacted with Prof. G. Hager (Yale) and Prof. D. Huttenlocher (Cornell) to acquire state-of-the-art visual tracking software. Eventually, we decided to use Huttenlocher's techniques (see Appendix C).

Recently, we have had several meetings with Prof. S. Rock (Aero-& Astronautics Dept., Stanford) to discuss cooperation on two projects: target finding and tracking with an autonomous helicopter (a project that Prof. Rock conducts with Boeing), and fish finding and tracking with an autonomous submarine (a project that Prof. Rock conducts with the Monterey Bay Aquarium Research Institute). Both projects are good application domains

for our techniques. In connection with these meetings, we have started the investigation of target finding in 3-D space with a new student (Cheng-yu Lee). We also have set up a small project for tracking fishes in an aquarium with two undergraduate students (Jian Bao and Stephen Sorkin).

1.6 Summary

In summary, our research demonstrates that, in addition to being useful for navigation, vision sensors can also be the main "effector" of mobile robots. It also shows that teams of mobile robots equipped with cameras can accomplish tasks that no single robot could accomplish, and that they can accomplish other tasks with much greater reliability and efficiency than any single robot alone.

Our research has produced a variety of new algorithmic techniques for motion planning with visibility constraints. With the convergence of a number of new technologies (telepresence, multimedia, virtual and augmented reality, etc), we believe that this area will attract much more interest in the coming years. We have also developed generic agent architecture for cooperating autonomous observers [Hayes-Roth et al, 1995; Pfleger and Hayes-Roth, 1996]

Our experimental prototypes integrate techniques demonstrating the interplay of several capabilities, including visual tracking, landmark-based navigation, collision avoidance, visibility analysis, motion planning, and motion control.

Chapter 2

Three-Dimensional Model Building

This chapter presents a detailed description of our progress on the model building problem, which was briefly discussed in Section 1.2.1. The task is to coordinate the motions of one or more robots to build a visual model of a new environment. This model combines 3-D information along with texture maps. Most of our work so far has been targeted toward model acquisition and construction. Since environments can be very complex, we have strived to design techniques to generate sparse, yet realistic, models. Our work on computing a motion strategy optimizing the number of sensing operations is still preliminary.

2.1 Introduction

Imagine that a collection of autonomous observers are dropped into a new environment. Their initial task is to build a visual model of this environment suitable for both graphic fly-through operations and physical navigation. The model we wish to build consists of (1) a representation of the 3-D geometry of the environment in the form of a triangular or quadrangular mesh, and (2) color texture maps for each element in the mesh. The automatic construction of such visual maps have many potential applications, independent of other functions that autonomous observers may offer (e.g.: military, architecture, archeology, etc.).

Three-dimensional model construction has been a research focus within the computer vision and graphics communities since range-data acquisition hardware became an accessible experimental option to even moderately funded research groups, and range-data files have been made available via FTP. However, most of the work related to 3-D model construction has been done within the framework of computer graphics with fixed range-data acquisition platforms.

IO Map Building Subsystem

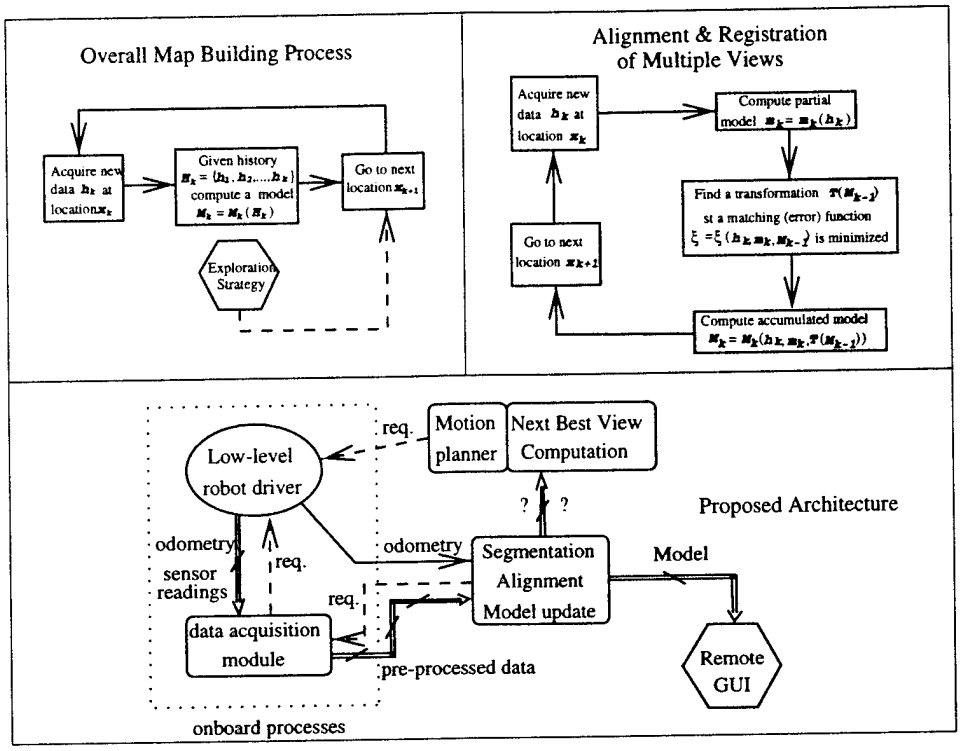


Figure 2.1: Software architecture for model building

Model construction using mobile robots poses particular issues that are not commonly seen in fixed range-data acquisition platforms. For instance, robot odometry is far less precise than the positioning sensors of fixed systems, which makes the alignment of successive views and the computation of texture maps significantly more complicated, especially in a multi-robot scenario where some robots acquire range images while others obtain the color information (distributed sensing, see [74]). Another issue is the size of the model. While fixed range-data acquisition platforms are used to build representations of relatively small objects, autonomous observers should be used to generate models of large and complex environments; moreover, in most cases we will want to send these models to remote locations using a limited-bandwidth network. Therefore, representations need to be as compact as possible.

We have studied an approach to model construction which yields the software architecture depicted in Figure 2.1. In this approach the model building process is decomposed in four phases:

1. **Model acquisition.** We wish to use the visual model to perform virtual fly-through

navigation through the environment at a fast image rate. Hence, the size of the 3-D mesh underlying this model is a crucial issue. One may precompute the visibility structure of the environment in order to avoid overwhelming the Z-buffer with many unecessary triangles. Instead, one may generate relatively coarse representations. Although these two approaches can be combined, our research has specifically explored the second. We have developed techniques to build and simplify sparse meshes.

2. **Texture mapping.** Once a geometric model has been obtained, we map color information given by color cameras into this model. Here, it is important to generate no or very few artifacts. To achieve this, we first establish correspondence between a few points in the geometric model and those in a camera image, and solve a minimization problem to recover the transformation that maps each point in the model onto the image. We then use this transformation to find the correspondence between arbitrary points in the model and those in the image in order to define the texture map for the reconstruction.

3. **Fusion of multiple views.** Several sensing operations, performed by the same observer moving at successive locations, or by several observers, give multiple 3-D/color images. The main issue to be solved is now the following: Given two 3-D/color images of two overlapping subsets of the environment, how can we fuse them into a single model? The issue is complicated by the fact that the localization of the observers may not be very precise, which requires the partial models to be geometrically aligned before fusion.

4. **Sensing strategy planning.** Sensing operations and fusion of partial models can be extremely time consumming. We must therefore try to minimize the number of such operations, by planning the best set of positions where the observers should perform the sensing operations. In the absence of prior information, we propose to derive a sensing strategy from a simple 2-D model that is easier and faster to build than a 3-D one.

We now describe each phase in more detail.

2.2 Model Acquisition

Each autonomous observer is equipped with a laser-camera system for range-data acquisition. As described in detail in Appendix B, the range finder operates under the principle of triangulation. The laser projects a plane of light perpendicular to the orientation of a CCD camera equipped with an appropriate filter. Hardware on board the robot detects the pixel

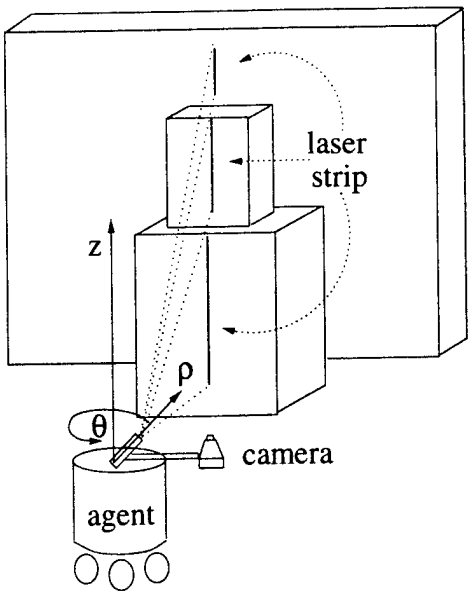


Figure 2.2: The range-finder takes a "cut" of the environment in one sensing operation.

of highest intensity value in each row of the image every time a new measurement acquistion takes place. One row is called a *scanline*, and the collection of all the intensity peaks for each scanline is called a *scan profile*. A scan profile is described by an array $\mathcal{A}$ of integers containing the column numbers of those pixels with the highest intensity value per scanline. Based on a sensor model (see Appendix B), the information contained in $\mathcal{A}$ is converted into a distance image. The calibration of the sensor is done prior to any model acquisition operation using a procedure described in Appendix B (subsection B.1.3).

A scan profile basically takes a "cut" of the environment as shown in Figure 2.2. As the robot rotates its turret, the plane of light sweeps over the surfaces in the scene, effectively acquiring scan profiles of the environment. During a sweep, the robot acquires a large collection of data points corresponding to those surface points visible to the camera when the illumintated by the laser plane of light during the acquisition process. Given this set of points, we desire to contruct a representation of the surfaces sampled by the sensor.

There has been considerable research in range-image segmentation, and good techniques have been proposed to recover surface information from range-data. Recently, the desire of teleoperating robots remotely through existing non-dedicated networks (such as the Internet) has sparked interest in fast on-line (incremental) reconstruction methods and in sparse-mesh representations (e.g., see [26]).

Our approach is based on the observation that, in a single sweep, range data is acquired with an implicit order due to the physics of the sensing device. To illustrate this point, consider the

robot-laser configuration shown in Figure 2.2. When the turret rotates counterclockwisely by $\theta < 2\pi$, the data collected is ordered in slices corresponding to increasing values of θ, and the sensor gets the data in each slice in descending order. Stating the problem in cylindrical coordinates (ρ, θ, z) leads to capture the environment surface as an array $\rho(\theta, z)$; a slice (column) of this array contains information about the changes in the distance when z varies, while successive slices describe the evolution of the surface when θ varies. We take advantage of this sorting to incrementally build a mesh description of the surface. We iteratively construct compact representations of the slices (or scan profiles) and we assemble these representations into surface meshes. In other words, the segmentation process yielding the mesh representation is done in two steps: first, along the vertical direction and then along the direction of the sweep.

2.2.1 Scan Profile Segmentation

Each individual profile can be processed and segmented during the acquisition stage, greatly reducing the amount of information transmitted wireless from the robot by this step alone.[1] To achieve this, we successively pre-filter, cluster, and segment the profile data.

The pre-filtering process is done to eliminate spurious peaks due to noise inherent to the sensing operation; this step is basic signal processing. Clustering groups the data traced from distinct surfaces into different groups, while segmentation builds a representation of each cluster using geometric primitives. The simplified process is described by the schematic shown in Figure 2.3.

Clustering

The goal of clustering is to separate contiguous groups of points in a scan profile that lie on surface patchs which, from the sensor perpective, are different. Refer to Figure 2.4. If we had a sensor with infinite resolution, points of discontinuity in the curve $r(\theta, z)$, at the current fixed θ, would only arise where the continuous light strip projected on the environment surface is sectioned or where occlusions occur along the line of sight of the camera. The question is how to break a scan profile that has finite resolution.

A simple procedure is to consider every two consecutive points in the scan profile and compute the distance between them. If this distance is below a certain threshold, then we assume that the two points are in the same surface patch and we include them in the same cluster.

[1] Our mobile robots communicate to the local network using radioethernet bridges. The amount of real-time processing and information sharing that can be outsourced to other computers is limited by the communication bandwidth.

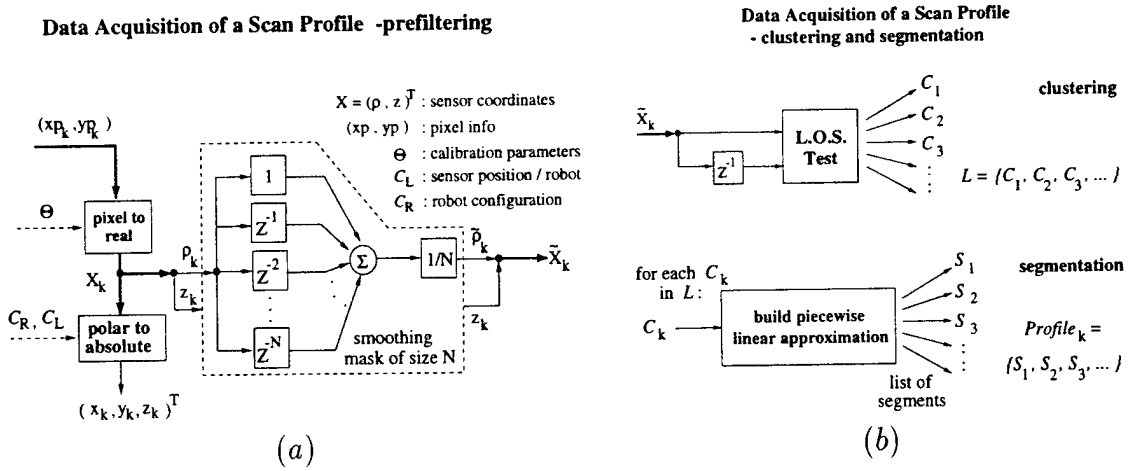


Figure 2.3: Pipeline for scan profile segmentation. (a) Initial data acquisition and prefiltering. (b) Clustering into distinct groups and conversion of each cluster into a list of segments.

Otherwise, we create a new cluster. One drawback of this procedure is that points far from the sensor are not scanned with the same resolution as those which are nearby. Hence, for a particular threshold value, we may cluster points located in the same surface patch into distinct groups; had this patch been located closer to the sensor, the scanned points would have been clustered together, instead. The converse could also happen if we tighten the threshold value. Although such a tradeoff is inherent to thresholding, we will like the generated clusters to be less sensitive to the choice of the threshold.

A better clustering approach is to use the *line of sight* test (LOS). This test (illustrated in Figure 2.4) uses the distance between successive points in the direction of the sensor line of sight, instead of the absolute distance. The LOS test starts by calculating $\hat{r}$, the unit vector describing the direction of the line of sight at a midpoint between two consecutive samples. Then, given s (the vector between samples), it computes $\mid s \cdot \hat{r} \mid$ and compares this distance to a threshold.

Piecewise Linear Approximation of Scan Profiles

We now represent each cluster of points as a series of straight-line segments.[2] We have developed a technique that generates this representation in linear time. It first transforms the point set into a more favorable coordinate system, then fits the point set with the weighted

[2]More complex primitives could be used here, if needed.

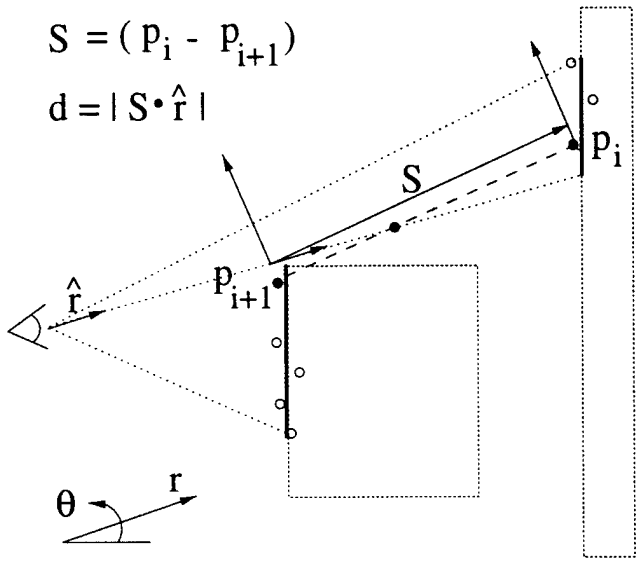


Figure 2.4: Line-of-sight test: The separation between points in the direction of the sensor's line of sight is used to determine if a pair is in the same cluster.

sum of the first n Chebyshev polynomials, and finally uses this continuous approximation as a noise-free representation of the surface being segmented. This technique is quite robust to noise in the measurement process.

This technique and another one (called successive pivoting) are described in detail in Section B.2 of Appendix B.

2.2.2 Profile Assembly

Profile segmentation produces a vector of segmented slices $L[i]_{i=1}^{n}$ as output, where every element of the vector is a list of clusters, i.e., $L[i] = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$. A cluster is itself a list of line segments, i.e., $\mathcal{C}_j = \{s_1, s_2, \ldots\}$. The vector $L[\cdots]$, called the *profile vector*, represents the environment surface sampled by the sensor in a single sweep.

The profile vector is converted into a mesh structure using the following algorithm:

Algorithm *Assemble mesh surface from segmented profiles*
Input:

1.- a vector of segmented profiles $L[i]_{i=1}^{n}$

2.- a predicate `bool Match(`$\mathcal{C}, M$`)` to establish if a sequence of segments $\mathcal{C}$ matches the partially-built mesh M

Output: a list of surface meshes $\mathcal{M} = \{M_1, M_2, \ldots,\}$, where $M_i = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ is a list of successive surface slices (list of segments) along the sweep direction

1. for $k = 1$ to n do
   while $L[k]$ not empty do
   extract (with deletion) the first element $\mathcal{T}$ from the list $L[k]$,
   let $M = \mathcal{T}$ and $i = k$,
   while $(i < n)$ do
   let $i = i + 1$, and `flag = false`
   for every element $\mathcal{C}$ in list $L[i]$ do
   if `Match(`$\mathcal{C}, M$`)`
   append $\mathcal{C}$ to surface M,
   remove element $\mathcal{C}$ from $L[i]$,
   let `flag = true` , and break.
   else continue.
   if `flag` is `false` break.
   else continue.
   if M has more than one element append to $\mathcal{M}$.
2. Filter list $\mathcal{M}$ to eliminate extremely small or thin meshes.
3. Make meshes uniform.

To work efficiently, the algorithm depends on a good definition of the predicate `bool Match(`$\mathcal{C}_i, M$`)`. The predicate must be accurate and robust to noise; its computational complexity must not explode with the size of M. We have experimented with different predicates. Although we had obtained good results with some of them, optimal predicate selection is still an open question.

2.3 Texture Mapping

Given a geometric model specified in a coordinate system $\mathcal{F}_A$ and the image data specified in the image coordinate system $\mathcal{F}_C$ (2.5), we would like to find the transform T that associates every point in the model with a point in the image, in order to map the image onto the geometric model.

Theoretically, a careful calibration procedure would solve our problem. Calibrating the

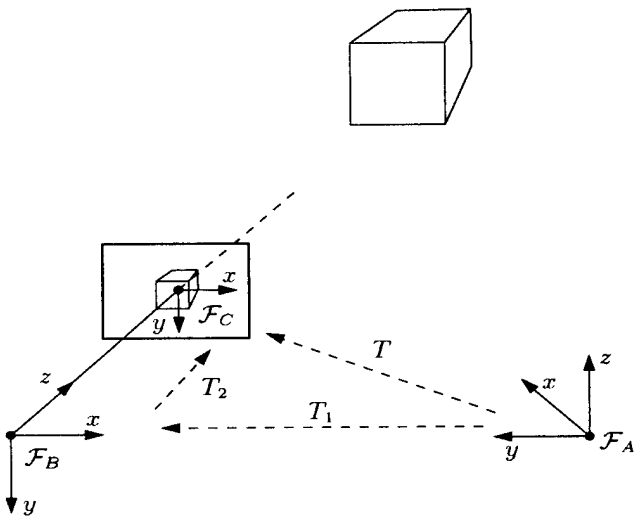


Figure 2.5: Geometric relationship between the RAS coordinate system $\mathcal{F}_A$ and the camera coordinate system $\mathcal{F}_B$. Transformation T maps a point specified in $\mathcal{F}_A$ to its coordinates in the image plane of the camera.

relative position and orientation of the camera's coordinate system $\mathcal{F}_B$ with respect to $\mathcal{F}_A$ gives the transformation T_1. Calibrating the camera gives the projective transformation T_2 that maps the coordinates of a point in $\mathcal{F}_B$ to its coordinates in the image coordinate system $\mathcal{F}_C$. Composing T_1 and T_2 provides the required transformation T. However, this is not a very practical approach for a mobile robot system. The relative placement of the range finder and the color camera are likely to change frequently as the robot is reconfigured for different tasks. Lenses and other components of the range finder and the camera may also change due to different task requirements, which would often result in time-consuming re-calibration. In addition, imperfections in the system make it virtually impossible to have a precise mapping over the entire range of operations.

2.3.1 Related Work

Our texture mapping problem is related to motion estimation and camera calibration for stereo matching. These problems has been addressed in the literature. A known result is that five pairs of points in non-degenerate positions are sufficient to recover the required projective transformation. In [52] it is shown that eight pairs determine a set of simultaneous linear equations to provide a unique closed-form under general conditions. A similar result

was presented in [71], showing that seven points are sufficient, barring degenerate cases. A method adequate for noisy data is the eight-point method presented in [75].

The above algorithms need pairs of corresponding points as inputs. Instead, feature-tracking algorithms based on image intensity can use cross-correlation [77, 72] or sum-of-squared-differences [53, 11] as similarity measures. These algorithms run a numerical method to optimize the chosen measure. Shi and Tomasi [67] presented a Newton-Raphson technique that works for a wide variety of similarity measures.

2.3.2 Our Algorithm

We propose to take the transform $T_0 : \mathcal{F}_A \rightarrow \mathcal{F}_C$, obtained from calibration as an initial estimate and compute a better estimate $\hat{T}$. The following procedure is used to obtained $\hat{T}$, where a quadrilateral mesh surface (the result of a previous segmentation procedure on the range-data) is used as input.

Algorithm *FindTransformation*
Input: geometric model,
image,
initial estimate T_0 of the transform from $\mathcal{F}_A$ to the image plane
Output: a better estimate $\hat{T}$

1. Project selected features of the meshed surface onto the image plane of the camera using T_0 to get a binary image I_1.
2. Apply edge detection to the input image to get a new image I_2.
3. Establish correspondence for features in I_1 and I_2 to obtain pairs of corresponding points (X_i, X_i'), where X_i is a point in the geometric model and X_i' is a point in the input image.
4. Solve the minimization problem

$$\min_T \sum_i |TX_i - X_i'|^2,$$

where T is a transform from $\mathcal{F}_A$ to $\mathcal{F}_C$.

In Step 1, the features selected for projection are the corners of the surface, that is, places on the surface with high curvature. At the corners, the surface normal changes sharply, which often results in sharp image intensity changes. Given a mesh surface, the curvature at mesh points can be easily estimated by finite difference methods.

In Step 3, we find corresponding features in I_1 and I_2 based on image intensity. This can be done either manually, or automatically by a template matching procedure based on cross-

correlation, sum of squared differences, or Hausdorff's distances.

Once we have $\hat{T}$, we define our texture mapping procedure as follows:

Algorithm *MapTexture*
Input:
A set S of mesh points whose corresponding points in the image have been found by explicit matching in *FindTransformation*,
Transform T
Output: a texture mapping function
1. For each mesh point $X_i \in S$, its corresponding texture coordinates are U_i.
2. For each mesh point not in S, compute its texture coordinates by using T.

T is used to map only points not in S because we feel that correspondence established by explicit matching is more reliable and yield better reconstructed images.

2.3.3 Experiments

The images in Figure 2.6 show experimental results on three different scenes. The top row shows a picture of each scene, the second row shows the 3-D model with texure mapped with an inaccurate initial guess of the transform. The third row shows a distorted texture map, the result of mapping explicitly identified correspondence pairs. The last row is the result of mapping the rest of the mesh using the updated transformation estimate $\hat{T}$.

One potential problem in *FindTransformation* is that the minimization procedure may get trapped in a local minimum. We never observed this problem, however, despite the fact that in all our experiments, the initial estimate T_0 supplied to *FindTransformation* contained severe errors.

2.4 Fusion of Multiple Views

Several sensing operations, performed by one or several observers, result in multiple partial models that must be fused into a single model.

Model fusion is complicated by the fact that the localization of the observers may not be very precise, which requires the partial models to be geometrically aligned before fusion. This is the *alignment* step. Aftewards, distinct models are combined together by *mesh fusion*, which merges the mesh representations of different views into a single quadrilateral mesh structure. These two operations operate over the pair (m_k, M_{k-1}), which is composed of the most recently acquired partial model (m_k) and the current accumulated model (M_{k-1}).

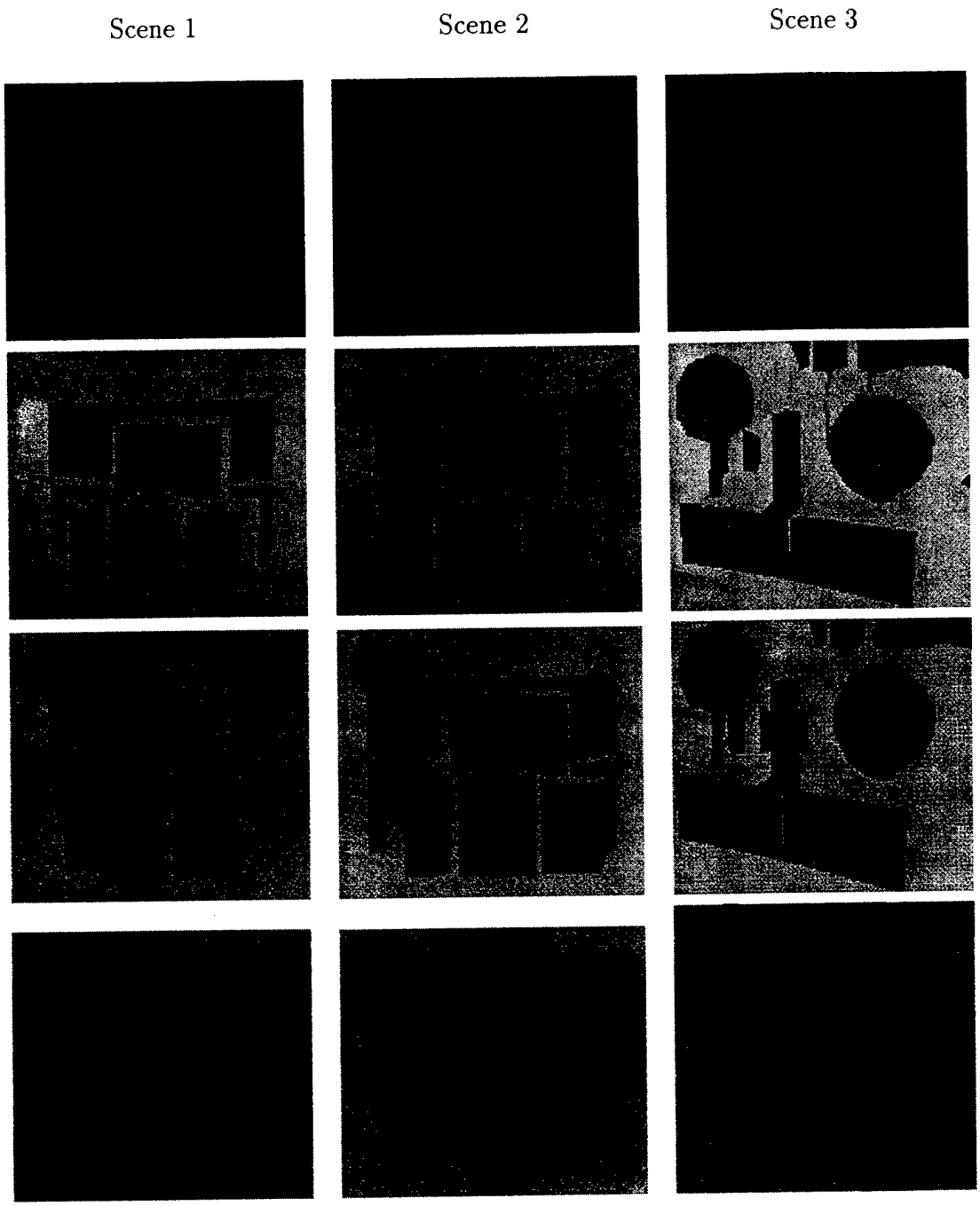


Figure 2.6: Some examples of reconstructed scenes with texture mapping.

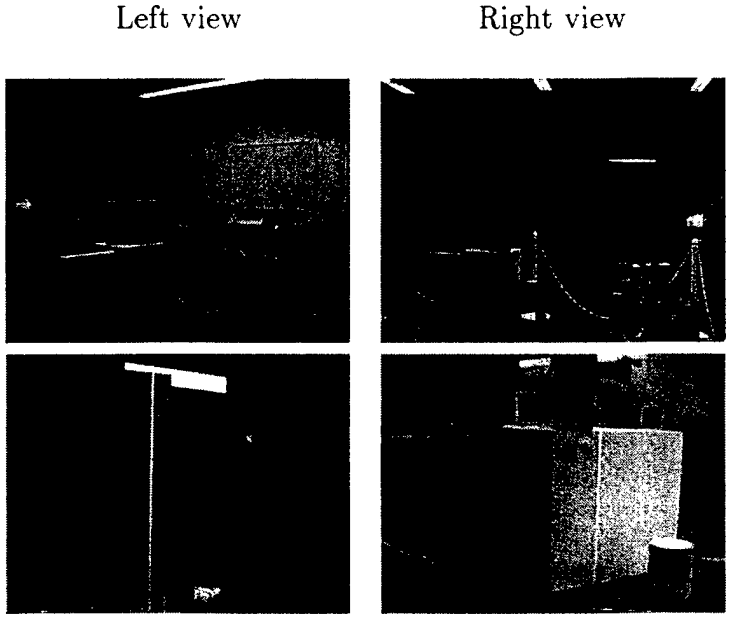


Figure 2.7: A scene is captured from two different viewpoints (left and right columns). The bottom row is the scene as observed from the range-finder CCD camera.

2.4.1 Alignment

Consider the situation shown in Figure 2.7, where a pile of boxes is scanned from two different viewpoints. Since distinct surfaces are captured during the model acquisition stage, two different set of surface meshes are built, each one representing a portion of the scene.

Geometric features captured by both viewpoints should be present in both partial models. Ideally, when transformed into a global coordinate system, the models should be superimposed at the common features. Good superimposition, unfortunately, is rarely observed in practice. A model constructed in a single scan sweep can be accurately expressed in the robot's local coordinates. To convert it into a world coordinate system, however, the robot's position must be known precisely, which is often not the case. We usually get a model superimposition like the one shown in Figure 2.8. The alignment step (also called registration) attempts to recover the transform T that characterizes the superimposition error.

Our aligment procedure, shown in Figure 2.9, computes the transform T such that the pair m_k and $T(M_{k-1})$ minimizes some error criterion. The algorithm is the following:

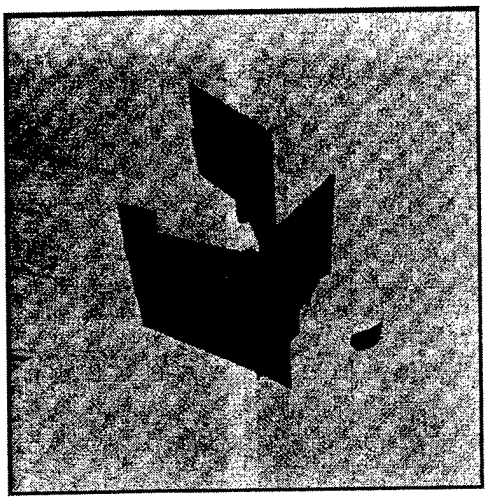


Figure 2.8: Localization errors will yield imperfect superimposition of two reconstructed views.

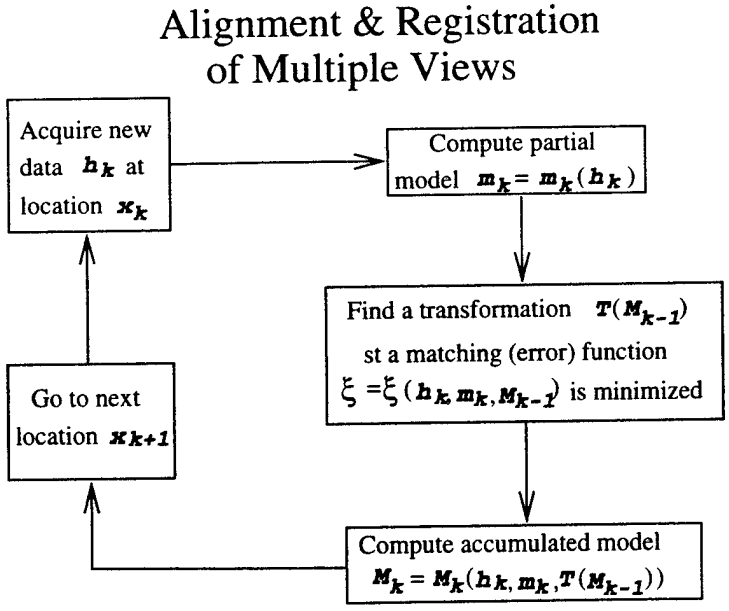


Figure 2.9: Alignment algorithm flow chart.

Algorithm *Align the current model with the last partial view*
Input:

1.- new measurements h_k taken during the last scan sweep k
2.- a model acquisition procedure to obtain the mesh representation m_k from h_k (see Section 2.2)
3.- the accumulated model M_{k-1}
4.- a perfomance criteria $\xi = \xi(h_k, m_k, M_{k-1})$

Output: a transformation $T(M_{k-1})$ that minimizes

$$\xi(h_k, m_k, T[M_{k-1}])$$

1. Compute the partial mesh reprsentation m_k given the acquired information h_k (see Section 2.2)
2. Solve the minimization problem

$$T^* = \arg\min_T \xi(h_k, m_k, T[M_{k-1}]). \qquad (2.1)$$

The optimization step of the algorithm can be done using any existing minimization procedure (see [63]). More critical, however, is the selection of the perfomance criteria ξ. We propose three alternatives, each based on three possible measures of mesh separation: mesh-to-mesh overlap, point-to-mesh separation, and contour-to-contour overlap.

Mesh-to-Mesh Overlap

One way to define ξ is to compute the overlap of each quadrilateral in mesh m_k with mesh $\bar{M} = T(M_{k-1})$. A quadrilateral overlap is computed by finding the projection of its area into $\bar{M}$, and multiplying it by a decreasing function of distance $w(r^{-1})$ (see Figure 2.10). The error ξ will be the negative sum of overlaps of all quadrilaterals in m_k with $\bar{M}$. Hence, when we execute step 2 in the alignment algorithm we will in fact maximize the degree of overlap beween m_k and $\bar{M}$.

The result using mesh-to-mesh overlap is very satisfactory. The views superimposed in Figure 2.8 are aligned as shown in Figure 2.11. The minor artifacts appearing in the figure are the result of quantization due to integer mesh representations.[3]

Mesh-to-mesh overlap has several drawbacks, however. It does not guarantee alignment, even if the optimization algorithm is perfect, and it is computationally expensive. In our experiments, alignment is by far the most consuming procedure of the whole model building process, exceeding mesh construction and mesh fusion in by orders of magnitude.

[3]There is no need to use integer representations. We selected them to ease coding.

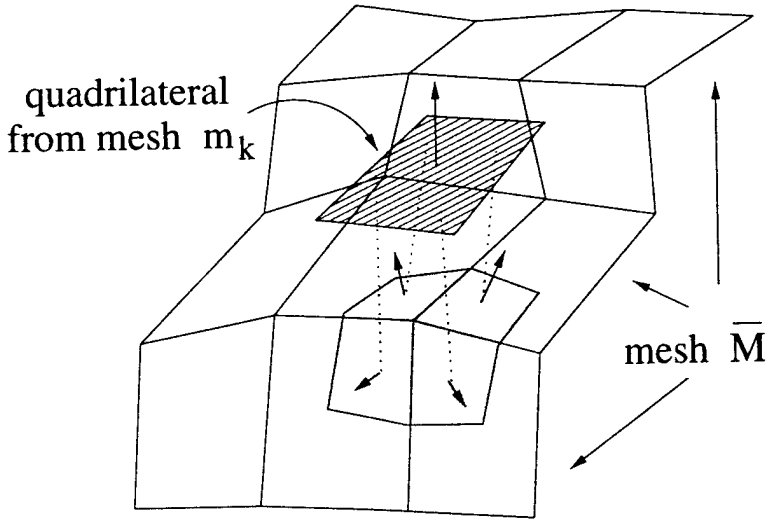


Figure 2.10: Overlap of a quadrilateral from m_k with mesh $\bar{M} = T(M_{k-1})$.

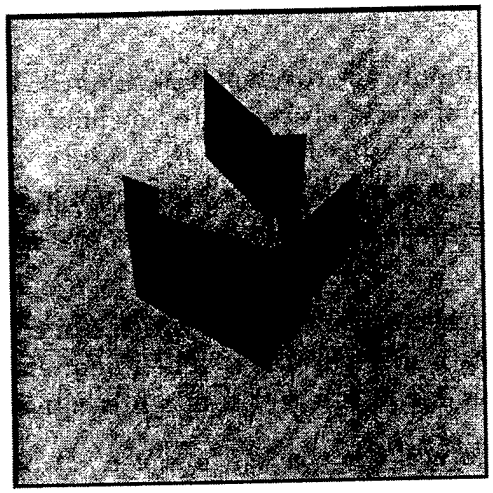


Figure 2.11: Two views are matched using mesh-to-mesh overlap .

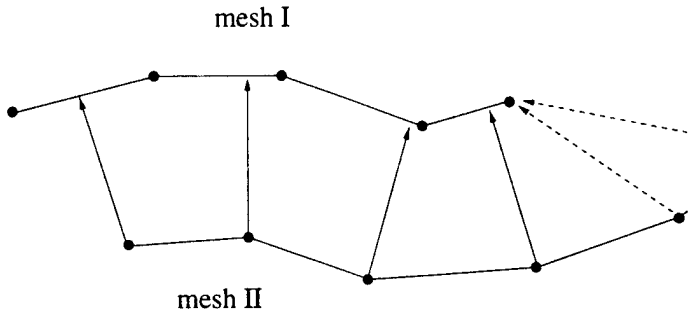


Figure 2.12: Correspondence points for the ICP algorithm. Pairs too far apart and points in the mesh boundary should be avoided as they drag the mesh in a direction opposite to the majority of correspondences (from Turk and Levoy 94).

Point-to-Mesh Separation

This approach is based on the *iterated closed-point algorithm* (ICP) proposed by Turk and Levoy in [73]. The basic process is the following: Compute the nearest position in mesh m_k to each vertex in mesh M_{k-1}, find the transformation that minimizes the sum of squared distances between pairs, and iterate the whole process until convergence.

The idea behind ICP is that, by computing the points in m_k nearest to the vertices of M_{k-1}, we will have pairs of points that correspond roughly to the same location in the true surface. Hence, these pairs should be used to compute the degree of matching. After such problem is solved, we should repeat the process to allow the fact that the pairs were not exact matches.

Turk and Levoy recommend that points in the boundary of m_k should be discarded, as well as those pairs of points that are too far apart. The reasoning behind this can be understood from Figure 2.12. Points in the mesh boundary tend to drag a mesh in a direction opposite to the majority of correspondences. At the same time, pairs too far apart are likely to correspond to portions in $\bar{M}$ not scanned in m_k.

Given a number of pairs, the best rigid transformation in the least-squares sense can be computed in linear time using the closed-form solution obtained by Horn (see [32]). Horn's method finds the displacement vector δ and the rotation $\mathcal{R}$ such that

$$E = \sum_{i=1}^{n} \mid p_i - R(q_i - q_c) - \delta \mid^2$$

is minimized, where $(p_i, q_i)_{i=1}^n$ are the given pairs, and q_c the centroid of all the q_i's.

The ICP algorithm is superior in speed to mesh-to-mesh overlap techniques:

Algorithm *Iterated closed-point alignment*
Input:

1.- a partial model m_k

2.- the accumulated model M_{k-1}

Output: the transformation $T(M_{k-1})$ that alignsM_{k-1} with m_k

1. Find the nearest point on mesh m_k to each vertex in M_{k-1}.
2. Discard pairs that are too far apart.
3. Eliminate pair in which either point is on a boundary.
4. Find the rigid transformation that best aligns the pairs in the least-squares sense.
5. Iterate the process until convergence.

There two main problems with the ICP algorithm. One is extracting the correpondence pairs. If the mesh vertices are uniformly distributed, and its quadrilaterals limited in size, then it is possible to perform a uniform subdivision of the space based on a distance threshold. With the subdivision we may constrain the nearest-point search to a local neighborhood, outside of which we can guarantee the solution does not exist. However, for sparse meshes we do not have uniformity, and no hard bound on the quadrilaterals' size. The nearest-point query may then be expensive, although this issue is made less critical by the fact that a sparse mesh should not contain a very large number of quadrilaterals in the first place (except for highly curved objects). The second problem is determining the threshold beyond which we determine a point pair as too far apart. The threshold can be easily determined if we have a bound on the size of the quadrilaterals. Again, for a sparse mesh, this is not possible for general scenes. One must use heuristics to obtain the threshold value.

Contour-to-Contour Overlap

An extension of the mesh-to-mesh overlap technique is to use contours. It is based on the observation that the robot's position is given by three coordinates (x, y, θ). Mismatchs are due to errors in the estimation of these coordinates. So, T should consist of two translations T_x and T_y and a rotation T_θ. We can take a x-y cross-section cut of m_k and $\bar{M}$ at some height z, and use 2-D matching techniques to recover T^* entirely (see Figure 2.13). A better estimate of T^* can be computed by using more than one cut.

By using solving the problem in cross-sectional cuts, we fall into one of the classic problems in computer vision: template matching. Different techniques had been proposed in the literature to solve this problem. See [77, 72] for cross-correlation approaches, and [53, 11] for the use of sum-of-squared-differences as a similarity measure. A Newton-Rapshon approach presented in [67] can be used for a wide variety of similarity measures. If we do not think orientation errors are present ($T_\theta = 0$), we can use the fast techniques proposed by

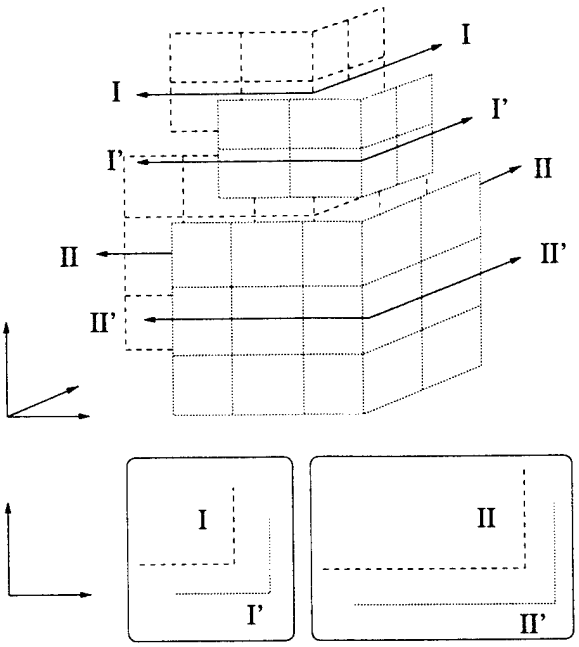


Figure 2.13: By matching cross-section cuts of two meshes, we can recover the transformation $T^* = (T_x, T_y, T_\theta)$ that aligns the views.

Huttenlocher to recover T_x^* and T_y^* (see [37] and [36]).

Alignment operations over cross-sectional cuts can be executed very quickly, so this approach looks very promising. We are currently in the process of evaluating its practical perfomance.

2.4.2 Mesh Fusion

After alignment, we need to fuse the mesh of m_k with the mesh of $T(M_{k-1})$. Consider the situation shown in Figure 2.14. Two meshes resulting from distinct views are shown together after alignment. The mesh regions with no overlap should be detached from the rest of the structure to form a new independent mesh. The overlapping information from both views should be fused together into a new middle mesh. In order to merge two models together, one must compute all the intersecting meshes.

The key function in the mesh fusion process is the predicate
`bool Match-Pair`$(a, b, left, right, compose)$, which returns `true` if meshes a and b intersect and `false` otherwise. If they do intersect, *right* and $left$ return the non-intersecting portions (residuals), and *compose* returns the composite (overlapping) mesh.

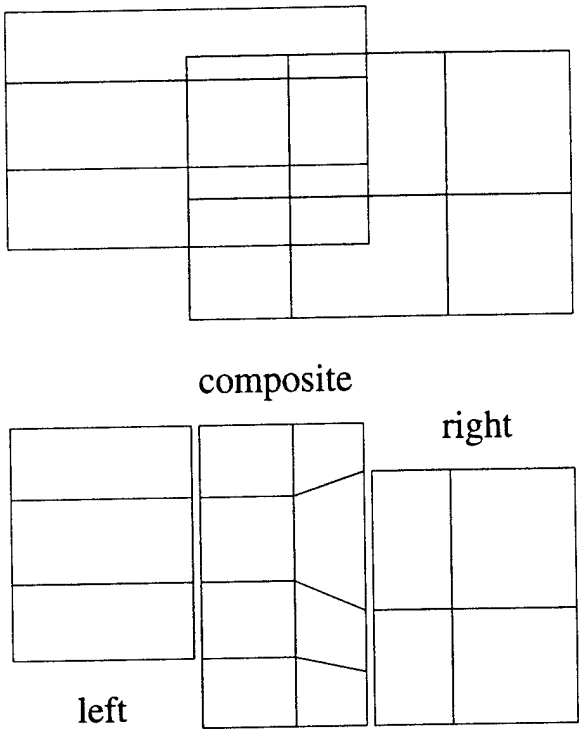


Figure 2.14: Merging two meshes together. If the structures intersect, the merging function returns the residuals *left* and *right*, and the composite mesh.

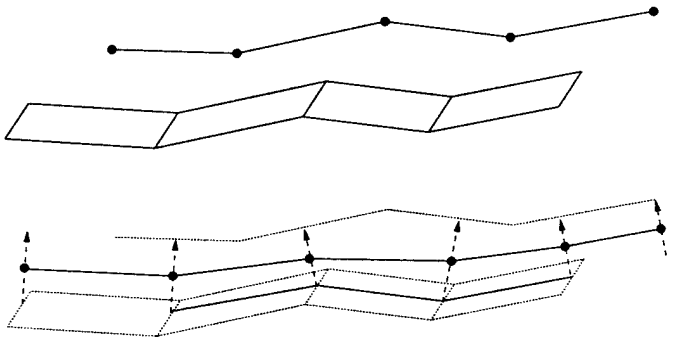


Figure 2.15: Projection of profile onto a mesh slab.

`Match-Pair(·)` runs by repeatedly executing the operation shown in Figure 2.15. This basic operation computes the projection of a piecewise linear curve onto a mesh slab. Midway between the projection and the original curve, a new segment sequence is constructed. The construction extends within the endpoints of both the original curve and the slab. The details of this operation are tediously complicated, and do not give much insight about the fusion process, so we omit the details.

The fusion algorithm operates over two models $\mathcal{A}$ and $\mathcal{B}$. Each model is composed of a list of quadrilateral meshes. In order to merge them together, we first define a function `bool Merge(`$b, \mathcal{A}, \mathcal{R}$`)` to merge a single mesh with a complete model:

Algorithm `bool Merge(`$b, \mathcal{A}, \mathcal{C}$`)` *Fuse a mesh with a model*
Input:
a mesh b
a model $\mathcal{A} = \{a_1, \ldots, a_n\}$
Output:
a list $\mathcal{C}$ containing all the composite meshes,
$\mathcal{A}$ returns containing its non-intersecting elements only,
returns `false` if there is no match

1. Let $\mathcal{C}$ and $\mathcal{R}_{aux}$ be empty
2. while (`true`)
   if $\mathcal{A}$ is empty
   Let $\mathcal{A} = \mathcal{R}_{aux}$
   return `false`
   Move the first item in list $\mathcal{A}$ to item *top*
   if (`Match-Pair(`*top*, b, *left*, *right*, *compose*`)`)
   if *left* residual was part of *top* append *left* to $\mathcal{R}_{aux}$
   else if (`Merge(`*left*, $\mathcal{A}$, *aux*`)`) append *aux* to $\mathcal{C}$
   else append *left* to $\mathcal{C}$
   if (`Merge(`*compose*, $\mathcal{A}$, *aux*`)`) append *aux* to $\mathcal{R}_{aux}$
   else append *compose* to $\mathcal{R}_{aux}$
   if *right* residual was part of *top* append *right* to $\mathcal{R}_{aux}$
   else if (`Merge(`*right*, $\mathcal{A}$, *aux*`)`) append *aux* to $\mathcal{C}$
   else append *right* to $\mathcal{C}$
   insert list $\mathcal{R}_{aux}$ to top of $\mathcal{A}$
   return `true`
   else append *top* to $\mathcal{R}_{aux}$

The algorithm is recursive. It checks the intersection of the top of $\mathcal{A}$ with b, storing the non-intersecting residuals in the auxiliary list $\mathcal{R}_{aux}$. If *top* does not overlap with b, then it

is added to the auxiliary list of residuals. When *top* is matched with b, the *left* or *right* residuals may intersect further with the rest of $\mathcal{A}$ if they are residuals of b; hence, the `Merge`$(\cdot)$ function is invoked again. The compositions computed in the recursive calls are added to the list of composites $\mathcal{C}$. The function returns after joining the auxiliary list of residuals to the returned non-overlapping reminder of $\mathcal{A}$.

Once we are able to merge a mesh to a complete model, model-to-model fusion is achieved by the following procedure:

Algorithm *Merge models $\mathcal{A}$ and $\mathcal{B}$*
Input:
a model $\mathcal{A} = \{a_1, \ldots, a_l\}$
a model $\mathcal{B} = \{b_1, \ldots, b_m\}$
Output: the fused model $\mathcal{M} = \{m_1, \ldots, m_n\}$
1.
2. Let $\mathcal{M}$ be empty
3. for every surface mesh $b_i \in \mathcal{B}$:
   if (`Merge`$(b_i, \mathcal{A}, \mathcal{C})$)
   append $\mathcal{C}$ to $\mathcal{M}$
   else
   append b_i to $\mathcal{M}$
   insert list $\mathcal{A}$ to top of $\mathcal{M}$

The models are fused by first merging every element in $\mathcal{B}$ with model $\mathcal{A}$, while appending the composites and non-intersecting b_i's to $\mathcal{M}$. Since a well-defined model contains disjointed meshes, two distinct elements of $\mathcal{B}$ do not have equal overlaps with $\mathcal{A}$. Hence, after going through the whole cycle, whatever is left in list $\mathcal{A}$ are the the non-overlapping pieces of the first model, which are added to $\mathcal{M}$.

The results using computed by the merge algorithm are good. The views superimposed in Figure 2.8 are fused as shown in Figure 2.16. Both views are now fused into a single consistent mesh, ready for color texture mapping.

2.5 Sensing Strategy Planning

Most of the sensing strategies appearing in the literature address the automatic exploration problem as sequence of next-best-view problems (e.g., [3, 17, 54, 62, 76]). Existing techniques to solve the next-best-view problem are not ideally suited to our application for two reasons. First, solving for the next-best view is a *local* planning problem, and the resulting sequence of views may yield too many sensing operations. In our case, each sensing

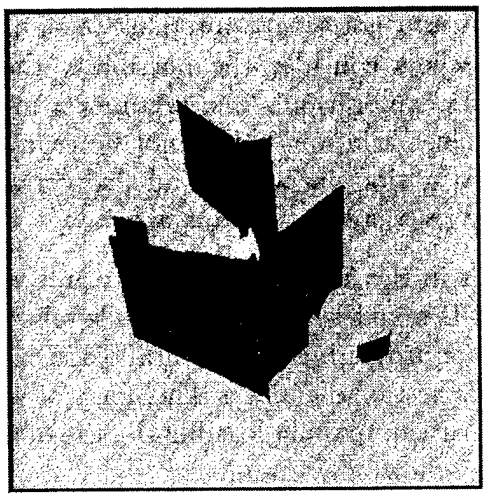


Figure 2.16: Two views are merged into a single consistent mesh.

operation is expensive; it requires acquiring 3-D and texture data and merging this data with the current model. Therefore, it is desirable to keep sensing operations to a minimum. Second, sensor localization is not perfect if the measurement device is mounted on a robot –odometric errors between successive views may cause misalignments resulting in incorrect models. Model construction using measurements from multiple views requires 3-D data to be aligned (through a partial matching operation) before merging, but this problem becomes more difficult to solve when the number sensing operations grow.

This remarks led us to address the model building problem in a different way. First, given a two-dimensional layout of the environment, we apply an art-gallery algorithm [58] (also called museum-guard problem) to compute a small number of positions such that if an observer was located at each one of these positions, the observers would collectively see the entire 2-D environement. Next, we move the observers along a path passing through each of the guard positions; at each position the agents perform a 3-D/texture sensing operation. Concurrently, the partial models provided by these operations are combined into a single consistent global model. Finally, remaining "holes" in the global model can be filled by adding a few local sensing operations using next-best-view techniques.

In the next subsections we present some of the issues present in our scheme.

2.5.1 2-D Map Construction

A viable strategy for automatic model construction is to consider the environment to be a 2-D workspace. Although it is reasonable to consider the mobile robot translations to be

restricted to a 2-D surfaces, it may not seem appropiate to assume the explored environment as planar. But some researchers consider the possibility that a 2-D layout may give a good strategy for the full 3-D exploration. The arguments in favor are that for some 3-D enviroments the vertical surface variations might not be crucial, that maybe a 2-D map is all what is required (as for navigation purposes), and that certain types of occlusions cannot be resolved by a range-finder restricted to a plane anyway.

The 2-D model must be built in the first place, without preliminary knowledge of the environment. We can acquire a 2-D map by letting the mobile robot navigate in the environment and using a simple range sensor projecting a horizontal plane of light (sonars can also be used instead of, or in addition to, the range-finder). This form of sensing is fast, so the number of sensing operations is not critical, making feasible the use of a next-best-view technique to drive the robot around the unknown environment.

Several strategies had been proposed to solve 2-D exploration. The work in [16, 42] approach the problem from a computational geometry perspective. They attack the problem of on-line search of polygons. In [42] a type of polygon is introduced for which a $(1+\sqrt{5})/2$-competitive search is possible (the worst-case cost - or distance - is at most $(1+\sqrt{5})/2$ times worse than the optimal strategy given complete knowledge of the geometry). In [16] general polygons are considered, and an algorithm for the incremental exploration of the workspace is provided. Approaches to the problem based exclusively in computational geometry issues give results that are clean, precise, and provable, but usually do not consider physical limitations of any type and assume perfect sensing. They are important, however, in that they provide an idea of the basic geometric problem, and establish bounds of how effectively we can solve it.

In [39] an algorithm for purposive sensing and motion for 2-D map building is presented. The authors consider real sensors (sonars), and collision-avoidance issues. The method consists on incrementaly building a topological graph (or roadmap) of the known environment computed so far, and updating this graph by connecting local structures extracted from the sensory data at each sensing location. The method has the advantage that, from a roadmap, the motion planning problem can be readily solved, so the proposed method solves both the exploration and the motion planning problem simultaneously. The graph approach results in new viewpoints that are reachable by the robot, making unneccesary a separate feasibility test.

2.5.2 Computing Sensing Locations from 2-D Map

As mentioned before, computing the ideal sensing locations from a 2-D map is closed related to the museum-guard problem. Most art-gallery algorithms assume the agents are equipped with an omnirectional and infinite-range sensor (assumption that simplifies the complexity

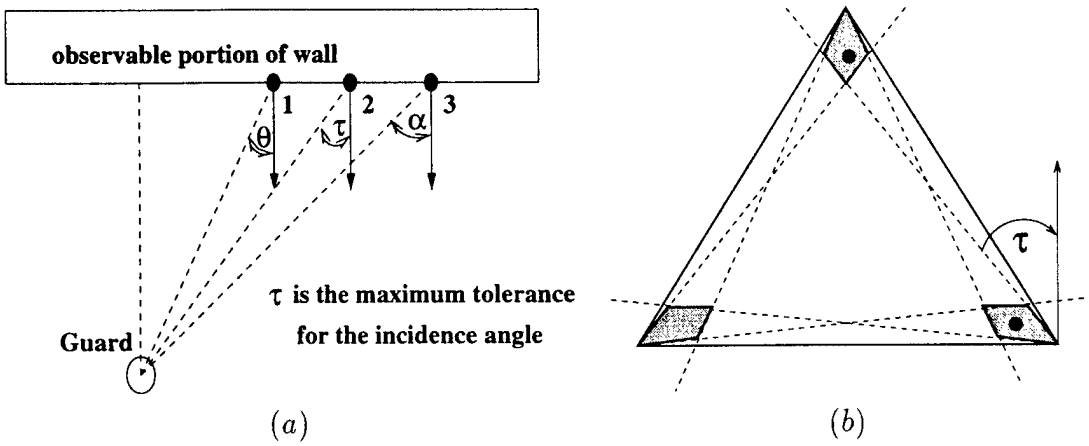


Figure 2.17: (a) Incidence contraint for a range-finder. Portions of the wall are seen only if $\mid \theta \mid \leq \tau$. (b) Complete coverage of even a simple triangle may be unattainable if incidence-angle restrictions are considered. The equilateral triangle may be covered by three guards, which should be located within the shaded areas.

of the problem). However, with real sensors, surfaces can be reliably seen only within some distance range and under non-grazing incidence angles. Taking such constraints into account yields new variants of the art-gallery problems.

Figure 2.17a shows how an incidence constraint limits the sensing range of the agent. The sensor is assumed to have a maximum tolerance of τ; i.e., when the incidence angle θ exceeds the value of τ the measurement is either unreliable or non-existent. This simple constraint already adds some interesting twists to the art gallery problem.

If an agent attempts to guard the inside of a polygon, corners may not be reachable. If α is the angle between two edges at a corner of a polygon, then the corner may be guarded only if $\alpha \geq 90^o - \tau$ (this is a direct consequence of the constraint $\mid \theta \mid \leq \tau$). In addition, for very large walls, only a section of size $2h\tan(\tau)$ is observable, where h is the minimum distance from the sensor to the wall. Hence, a wall of size l requires at least $\texttt{ceil}[l/(2h\tan\tau)]$ observations if the agent is constrained to within a distance h from the wall.

A remarkable consequence of condition $\alpha \geq 90^o - \tau$ is that not all triangles are scannable. Since the inner angles of a triangle add 180^o, then $180^o \geq 270^o - 3\tau$ which implies that $\tau \geq 30^o$. That is, if the maximum tolerance is less than 30 degrees, no triangle is scannable. If a triangle is scannable, it may even need three guard positions (see Figure 2.17b).

Our current research is focused on developing practical algorithms to solve the extended versions of the museum-guard problem arising in our automatic model construction strategy.

2.6 Discussion

We have described a set of implemented techniques to build a visual model of an environment. This model combines 3-D information (a quadrilateral mesh structure) and functions mapping color textures onto each quadrilateral mesh. This model can be used for virtual fly-through navigation on a graphic workstation, as well as for real navigation by mobile robots. A major goal of this research was to generate sparse, but still realistic mesh structures and to deal with imperfect localization of the observers. Future research should more throughly address the issue of computing motion strategies minimizing the number of sensing operations.

Chapter 3

Target Finding

This chapter presents a detailed description of our progress on the target finding problem, which was briefly discussed in Section 1.2.2. The task is to coordinate the motions of one or more robots that have omnidirectional vision sensors, to eventually "see" a target that is unpredictable, has unknown initial position, and is capable of moving arbitrarily fast. A visibility region is associated with each robot, and the goal is to guarantee that the target will ultimately lie in at least one visibility region. Both a formal characterization of the general problem and several interesting problem instances are presented. A complete algorithm for computing the motion strategy of the robots is also presented, and is based on searching a finite cell complex that is constructed on the basis of critical information changes. A few computed solution strategies are shown. Several bounds on the minimum number of needed observers are also discussed.

3.1 Introduction

Have you ever searched for someone in a building, possibly exploring the same places multiple times, while finally fearing that the person might have moved to a location already explored? Have you wondered how many searchers it would take to be able to guarantee that the person will eventually be located? This chapter presents a formal study of problems of this type, for which the task is to plan a motion strategy that will ensure that a target will eventually be "seen" in a workspace that is cluttered with obstacles that prohibit subsets of configurations of the searchers and also obstruct their visibility. The only assumption made about the target is that its motions are continuous. Obviously, the motion strategy must ensure that each portion of the workspace is in view at some point in time; however, the more difficult task is to prevent the target from "sneaking" into a region that has already been explored.

Several applications can be envisioned for problems and motion strategies of this type. For example, suppose a building security system involves a few mobile robots with cameras or range sensors that can detect an intruder. Stationary or limited degree-of-freedom camera bases could also be installed. A patrolling route can be automatically computed that guarantees that any mobile intruder will eventually be found. To optimize expenses, it would also be important to know the minimum number of robots that would be needed. Applications are not necessarily limited to adversarial targets. For example, the task might be to automatically locate another mobile robot, items in a warehouse or factory that might get moved during the search process, or possibly even people in a search/rescue effort. Such strategies could be used by automated systems or by human searchers.

Since the task is to *guarantee* that the target is found for all possible target motions, worst-case analysis will naturally be considered for modeling the target. In the analysis, the target will thus be termed an *evader*, although in an application the actual target might not be adversarial. Likewise, the robots that are equipped with vision or range sensing are termed *pursuers*, instead of observers. The pursuers and evader are modeled as points in the plane (alternatively general configuration-space representations could be used [46], but only 2-D configuration spaces are addressed in this chapter), and only continuous motions are permitted. Two interesting research issues follow from this general problem: 1) What bounds can be established on the number of pursuers needed to solve the problem, expressed in terms of the geometric and topological complexity of the free space? 2) Can a successful solution strategy be efficiently computed for a given problem? Our current progress on these two topics is discussed in this chapter.

The general problem is an extension or combination of problems that have been considered in several contexts. Pursuit-evasion scenarios have arisen in a variety of applications such as air traffic control, military strategy, and trajectory tracking. This has resulted in the formal study of general decision problems in which two decision makers have diametrically opposing interests. Classical pursuit-evasion games express differential motion models for two opponents, and conditions of capture or optimal strategies are sought [38]. For example, in the classical Homicidal Chauffeur game, conditions of inevitable collision can be expressed in terms of the nonholonomic turning-radius constraints of the pursuer and evader. Although interesting decision problems arise through the differential motion models, geometric free-space constraints are usually not considered in classical pursuit-evasion games. Once these constraints are introduced, the problem inherits the additional complications that arise in geometric motion planning.

A region of capture is often associated with a pursuit-evasion problem, and the "capture" for our problem is defined as having the evader lie within a line-of-sight view from a pursuer. Several interesting results have been obtained for pursuit-evasion in a graph [55, 61]. However, the visibility polygon along with motions in a free space add geometric information that

must be utilized, and also leads to connections with the static art gallery problems [58]. In the limiting case, art gallery results serve as a loose upper bound on the number of pursuers by allowing a covering of the free space by static guards, guaranteeing that any evader will be immediately visible. Far fewer guards are needed when they are allowed to move and search for an evader; however, the required motion strategies can become complicated.

3.2 Problem Definition

The pursuers and evader are modeled as points that translate in a 2-D bounded, Euclidean workspace that contains polygonal obstacles. See Figure 3.1 for illustrative examples. Let F represent the closure of the collision-free space (referred to as C_{valid} in [46]). All pursuer and evader positions must lie in F. Let $e(t) \in F$ denote the position of the *evader* at time $t \geq 0$. It is assumed that $e : [0, \infty) \rightarrow F$ is a continuous function, and that the evader is capable of executing arbitrarily fast motions. The initial position $e(0)$ and the path e are assumed unknown to pursuers.

Let $\gamma^i(t)$ denote the position of the i^{th} *pursuer* at time $t \geq 0$. Let γ^i represent a continuous path of the i^{th} pursuer of the form $\gamma^i : [0, \infty) \rightarrow F$. Let γ denote a motion) strategy, which refers to a specification of a continuous path for every pursuer: $\gamma = \{\gamma^1, \ldots, \gamma^N\}$ for N pursuers.

For any point, $q \in F$, let $V(q)$ denote the set of all points in F that are visible from q (i.e., the linear segment joining q and any point in $V(q)$ lies in F). A strategy, γ, is a *solution strategy* if for every continuous function $e : [0, \infty) \rightarrow F$ there exists a time $t \in [0, \infty)$ and an $i \in \{1, \ldots, N\}$ such that $e(t) \in V(\gamma^i(t))$. This implies that the evader will eventually be seen by one or more pursuers, regardless of its path. Let $H(F)$ represent the minimum number of pursuers for which there exists a solution strategy for F.

Two basic problems are considered:

1. Determine $H(F)$
2. For a given F, find a solution strategy, γ, using $H(F)$ pursuers

3.3 How Many Pursuers are Needed?

The minimum number of pursuers, $H(F)$, required to find an evader in a given free space F generally depends on both the geometry and topology of the free space. For example,

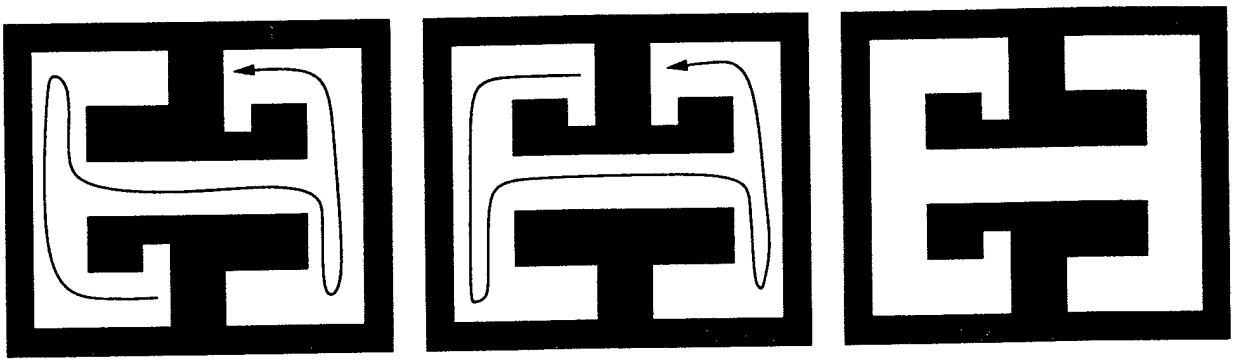


Figure 3.1: The rightmost example requires two pursuers, whereas the others require only one.

$H(F) \geq 2$ when F is multiply-connected (the evader can always hide behind a hole to avoid being seen by a single pursuer). Figure 3.1 shows examples that have subtle differences; however, $H(F)$ varies. In [68] a class of simple polygons called "hedgehogs" is identified for which a single pursuer suffices.

We have derived several bounds on $H(F)$ over certain classes of free spaces. It can be shown that for any simply-connected free space F with n edges, at worst $H(F) = O(\lg n)$, and for general free spaces with h holes, at worst $H(F) = O(h + \lg n)$ [31]. To obtain the first result, F can be recursively decomposed by placing pursuers on partitioning edges to prevent the evader from moving from one portion of F to another. A logarithmic number of pursuers can be systematically swept across partition edges to obtain the solution strategy. To obtain the second result, a linear number of line segments can be used to connect between holes and the exterior edges of F so that any continuous path that is not homotopic to a stationary path must cross one of the line segments. One pursuer can be placed on each segment to effectively reduce the problem to that of a simply-connected free space.

Lower bounds can also established which indicate problems that require at least some number of pursuers. To construct difficult problem instances, a well-studied problem from graph theory can be used. Let *Parsons' problem* refer to the graph-searching problem presented in [55, 61]. The task is to specify the number of pursuers required to find an evader that can execute continuous motions along the edges of a graph. Instead of using visibility, capture is achieved when one of the pursuers "touches" the evader. We have shown that for any instance of Parsons' problem on a planar graph, there exists an equivalent geometric instance [31]. The basic idea is to replace each edge in the graph by a thin corridor that has four bends. The key difference between the graph problem and the geometric problem is the power of visibility, which is essentially removed once four-bend corridors are used. By transforming difficult graph instances into geometric instances, it can be shown that exploring a tree of corridors of the type shown in Figure 3.2 requires $k + 1$ pursuers in which k is the height

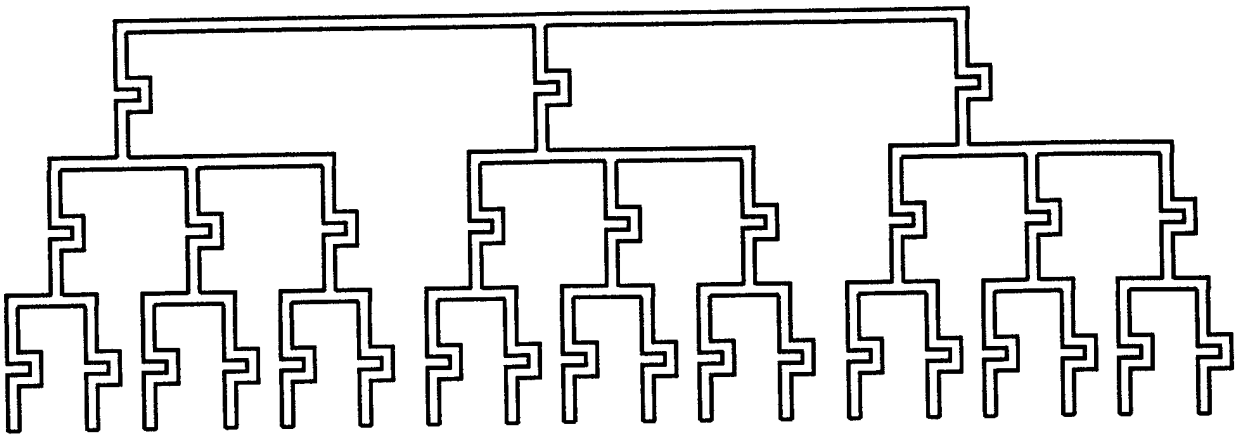


Figure 3.2: A ternary tree of bent corridors requires one pursuer per level (for this example $H(F) = 4$).

of the tree (thus establishing $H(F) = \Omega(\lg n)$) [31]. Examples can also be constructed that establish $H(F) = \Omega(\sqrt{h} + \lg n)$.

3.4 Computing a Solution Strategy

3.4.1 General Issues

In general, one would prefer a *complete* algorithm, which must compute a solution strategy for a given number of pursuers, if such a strategy exists. It is natural to compare the notion of completeness for this problem to completeness for the basic motion planning problem (i.e., the algorithm will find a collision-free path if such a path exists [13]). One important difference, however, is that the *minimum* number of pursuers is crucial, but does not have a correspondence for the basic path planning problem. A variety of simple, heuristic algorithms can be developed that require more pursuers than necessary (for example, triangulate the workspace, and place a static pursuer in each triangle). By building on some results from graph theory, it can be shown that the general problem of determining $H(F)$ for a polygonal environment is NP-hard [31]. The solutions can also be quite complicated (we have found examples that require clearing the same region $\Omega(n)$ times for n edges).

Because the position of the evader is unknown, one does not have direct access to the state at a given time. This motivates the consideration of an information space that identifies all unique situations that can occur during the execution of a motion strategy. Let a *state space*, X, be spanned by the coordinates $x = (x^1, \ldots, x^N, x^e)$, in which x^i for $1 \leq i \leq N$

represents the position of the i^{th} pursuer, and x^e represents the position of the evader. Since the positions of the pursuers are always known, let X^p denote the subspace of X that is spanned by the pursuer positions, $x^p = (x^1, \ldots, x^N)$.

It will be useful to analyze a strategy in terms of manipulating the set of possible positions of the evader. Any region in F that might contain the evader will be referred to as *contaminated*, otherwise it will be referred to as *cleared*. Let $S \subseteq F$ represent the set of all contaminated points in F. Let $\eta = (x^p, S)$ for which $x^p \in X^p$ and $S \subseteq F$ represent an *information state*. The information space is a standard representational tool for problems that have imperfect state information, and has been useful in optimal control and dynamic game theory (e.g., [2]), and in motion planning [4, 23, 47].

For a fixed strategy, γ, a path in the information space will be obtained by $\eta(t) = (\gamma^1, \ldots, \gamma^N, S(t))$ in which $S(t)$ can be determined from an initial $S(0)$ and the trajectories $\{\gamma^i(t') | t' \in [0, t]\}$ for each $i \in \{1, \ldots, N\}$.

We next describe a general mechanism for defining critical information changes. This is inspired in part by a standard approach used in motion planning, which is to preserve completeness by using a decomposition of the configuration space that is constructed by analyzing critical events. The next definition describes an information invariance property, which allows the information space to be partitioned into equivalence classes. A connected set $D \subseteq X^p$ is *conservative* if $\forall \eta$ such that $x^p \in D$, and $\forall \gamma : [t_0, t_1] \rightarrow D$ such that γ is continuous and $\gamma(t_0) = \gamma(t_1) = x^p$, then the same information state is obtained. This definition implies path invariance within a conservative cell [31]. Just as in the case of motion planning algorithms based on critical curves and noncritical regions [46], one can only consider sequences of cells in the search for a strategy while maintaining completeness. In our case, however, these cells exist in the information space.

3.4.2 The Complete Algorithm for One Pursuer

A conservative cell decomposition will be described that is based on critical changes in edge visibility. Suppose the pursuer is at a point $q \in F$. Consider the circular sequence of edges in the resulting visibility polygon. The edges generally alternate between bordering an obstacle and bordering free space. See Figure 3.3. Let each edge that borders free space be referred to as a *gap edge*. Consider associating a binary label with each gap edge. If the portion of the free space that borders the gap edge is contaminated, then it is assigned a "1" label; otherwise, it is assigned a "0" label indicating that it is clear. Let $B(q)$ denote a binary sequence that corresponds to labelings that were assigned from $q \in F$. Note that the set of all contaminated points is bounded by a polygon that must contain either edges of F or gap edges from the visibility polygon of the pursuer. Thus, the specification of q and $B(q)$

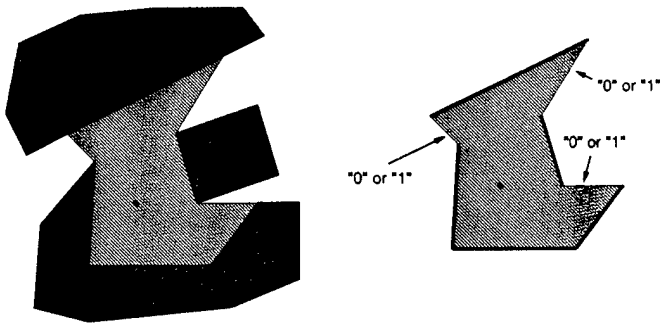


Figure 3.3: Edge labels can be used to encode the information state.

uniquely characterizes the information state.

Consider representing the information state using q and $B(q)$, and let pursuer move in a continuous, closed-loop path that does not cause gap edges to appear or disappear at any time. Each gap edge will continuously change during the motion of the pursuer; however, the corresponding gap edge label will not change. The information state cannot change unless gap edges appear or disappear. For example, consider the problem shown in Figure 3.4 which shows a single pursuer that is approaching the end of a corridor. If the closed-loop motion on the left is executed, the end of the corridor remains contaminated. This implies that although the information state changes during the motion, the original information state is obtained upon returning. During the closed-loop motion on the right, the gap edge disappears and reappears. In this case, the resulting information state is different. The gap label is changed from "1" to "0".

Hence, a cell decomposition that maintains the same corresponding gap edges will only contain conservative cells. The idea is to partition the free space into convex cells by identifying critical places at which edge visibility changes. A decomposition of this type has been used for robot localization in [30, 70], and generates $O(n^3)$ cells in the worst case for a simple polygon (which is always true if $H(F) = 1$). The free space can be sufficiently partitioned in our case by extending rays in the three general cases shown in Figure 3.5. Obstacle edges are extended in either direction, or both directions if possible. Pairs of vertices are extended outward only if both directions are free along the line drawn through the pair of points. This precludes the case in which one direction is cannot be extended; although edge visibility actually changes for this case, it does not represent a critical change in information. Our implementation uses the quad-edge structure from [29] to efficiently maintain the topological ordering of the conservative cells. Figure 3.8.a shows a computed example of the cell decomposition.

The next issue is searching the information space for a solution, which corresponds to specifying a sequence of adjacent cells. The solution strategy must take the form of a path that

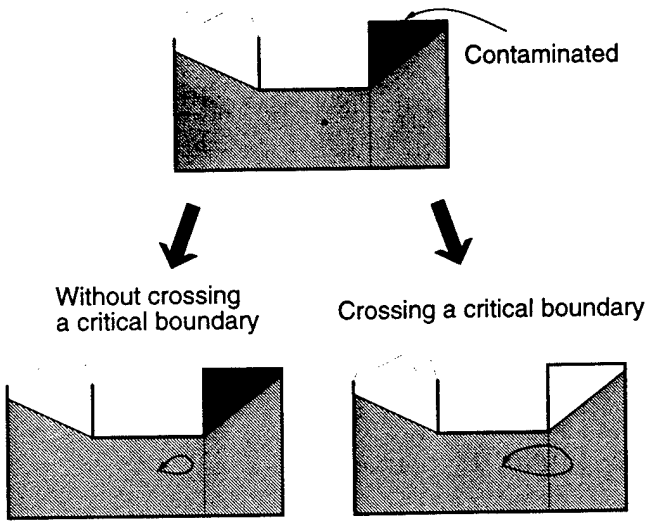


Figure 3.4: A critical event in the information space can only occur when edge visibility changes.

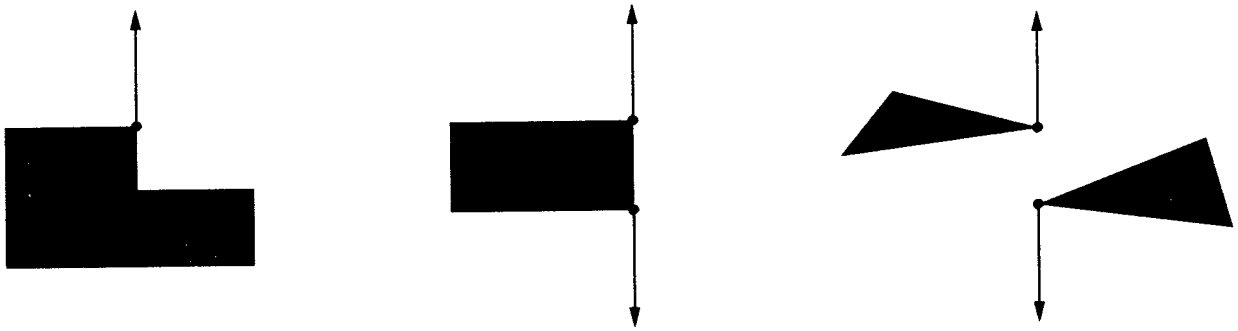


Figure 3.5: Ray shooting is performed for three general cases to form the edge-visibility cells.

maps into F. This can be constructed by concatenating linear path segments, in which each segment connects the centroids of a consecutive pair of cells in the sequence.

The cells and their natural adjacency relationships define a finite, planar graph, G_c, referred to as the *cell graph.* Vertices in G_c are generally visited multiple times in a solution sequence because of the changing information states. For each vertex in G_c, a point, $q \in F$, in the corresponding cell can be identified, and the labels $B(q)$ can be distinct at each visit. Initially, the pursuer will be in some position at which all gap labels are "1". The goal is to find any sequence of cells in G_c that leave the pursuer at some position at which all gap labels are "0".

A directed *information state graph*, G_I, can be derived from G_c, for which each vertex is

visited at most once during the execution of a solution strategy. For each vertex in G_c, a set of vertices are included in G_I for each possible $B(q)$. For example, suppose a vertex in G_c represents some cell D, and there are 2 gap edges for $B(q)$ and any $q \in D$. Four vertices will be included in G_I that all correspond to the pursuer at cell D; however, each vertex represents a unique possibility for $B(q)$: "00", "01", "10", or "11". Let a vertex in G_I be identified by specifying the pair $(q, B(q))$.

To complete the construction of G_I, the set of edges must be defined. This requires determining the appropriate gap labels as the pursuer changes cells. Suppose the pursuer moves from $q_i \in D_i$ to $q_j \in D_j$. For the simple case shown in the lower right of Figure 3.4, assume that the gap edge on the left initially has a label of "0" and the gap edge on the right has a label of "1". Let the first bit denote the leftmost gap edge label. The first transition is from "01" to "0", and the second transition is from "0" to "00". The directed edges in G_I are (q_i,"01") leads to (q_j,"0"), (q_j,"0") leads to (q_i,"00").

In the case of multiple gap edges, correspondences must be determined to correctly compute the gap labels. Consider the example shown in Figure 3.6 which illustrates the general cases that can occur. A gap edge from $V(q_i)$ corresponds to a gap edge from $V(q_j)$ if they share a vertex, and neither touch the extension of their common cell boundary. This case is shown in the upper left of Figure 3.6. In this case the binary label with be preserved when traveling directly from q_1 to q_2. The case is more interesting when gap edges touch the extension of the cell boundary, as in the lower portion of Figure 3.6. In general, all edges that touch the extension below the cell correspond to each other, and all edges that touch the extension above the cell separately correspond to each other. Transitions of this type essentially cause gap edges to be split or merged. There are two gap edges in the lower portion of Figure 3.6 while the pursuer is at q_1; however, there is only one gap edge when the pursuer is at q_2. In the transition from q_1 to q_2, if the gap edges at q_1 are labeled "0" and "0", then the corresponding gap from q_2 will be labeled "0". If either gap edge at q_1 is labeled "1", then the gap edge label from q_2 will be "1" (contamination spreads easily). In general, if any n gap edges are merged, the corresponding gap edges will receive a "1" label if any of the original gap edges contain a "1" label.

Once the gap edge correspondences have been determined, the information state graph can be searched using Dijkstra's algorithm with an edge cost that corresponds to the distance traveled in the free space by the pursuer. Unfortunately, the precise complexity of the complete algorithm cannot be determined because it is still open whether the problem even lies in NP. In the worst-case, examples can be constructed that yield an exponential number of information states, but it is not clear whether these information states necessarily have to be represented and searched to determine a solution (or to verify a solution).

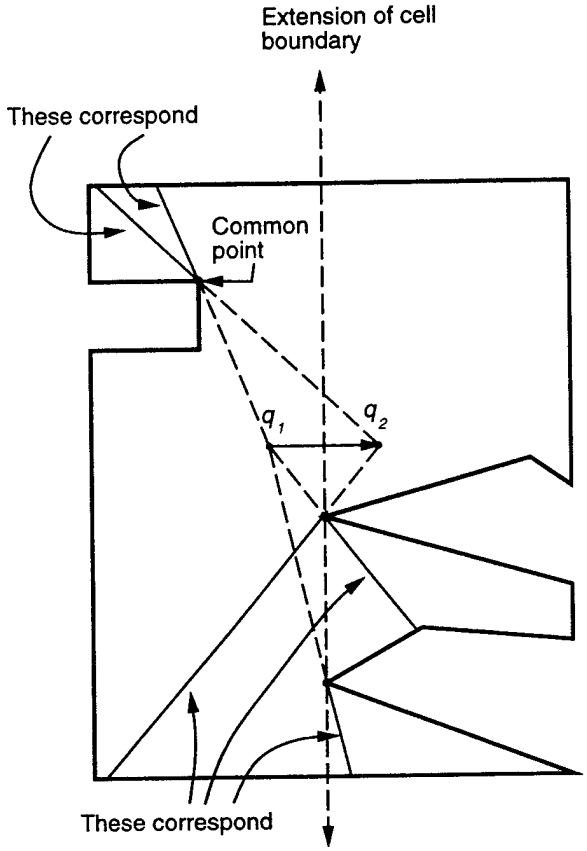


Figure 3.6: The correspondences between gap edges from different neighboring cells can be directly determined. The information states are updated when moving between cells by using this correspondence.

3.4.3 Coordinating Multiple Pursuers

In general, the conservative cell concept can be applied to yield a cell decomposition of X^p, which is the $2N$-dimensional space that encodes the positions of the pursuers; however, some of the cell boundaries are nonlinear algebraic manifolds. The nonlinear constraints significantly increase the implementation difficulty and add numerous cells which decreases practical efficiency.

In Section 3.5 we show some examples that were computed by coordinating multiple pursuers. We have developed a decoupled planning approach (losing completeness), which is inspired by typical approaches to multiple robot planning problems [46]. Suppose a problem cannot be solved by a single pursuer. The first step is to have the pursuer clear as much area as possible and stop. The fixed pursuer's visibility polygon partitions the free space into components

that can each by explored as a separate subproblem using the complete algorithm for a single pursuer. If each component can be solved by a single pursuer, then only two pursuers are needed in total (the same pursuer can be used for each component). In general, this type of search can be repeated recursively to coordinate more pursuers, until the problem is solved. In many cases, pursuers that are fixed during the clearing of one portion of the free space can eventually be moved to assist in another portion, further reducing the number of pursuers.

3.5 Computed Examples

3.5.1 Results from the Complete Algorithm

The complete algorithm was implemented in C++ and executed on an SGI Indigo2 workstation with a 200 Mhz MIPS R4400 processor. The computation times and other parameters for several examples are listed in Figure 3.7. The implementation uses the quad-edge structure from [29] to maintain the topological ordering of the conservative cells. The searching strategy is essentially Dijkstra's shortest path algorithm, where the distance is measured from the adjacent cell centroids. The solution is computed by traversing from cell centroids to cell centroids, causing the computed path for the pursuer to be jagged in most cases. In some applications, it might be appropriate to employ smoothing algorithms to the path to respect additional problem constraints.

Figures 3.8-3.12 show several computed examples. Due to a large number of conservative cells, Figures 3.10-3.12 are illustrated with the cell decompositions in separate diagrams from the solution diagrams. Figure 3.10 shows the hookpin example described in [68]. Note that the leftmost pin is recontaminated twice, and the pins are visited in the same order as mentioned in [68]. Figure 3.11 is an instance of the sequence described in Section 3.3 that requires a linear number of recontaminations. The region near the top of the figure is recontaminated 3 times. The final example generated a large number of conservative cells, which significantly increased computation time.

3.5.2 Results from the Greedy Algorithm

The greedy algorithm was implemented in C++ by extending the complete algorithm, and was also executed on an SGI Indigo2 workstation with a 200 Mhz MIPS R4400 processor. Three computed examples are shown in Figures 3.13-3.15. The total computation times for these examples were 0.21, 1.54, and 4.12 seconds, respectively. Figure 3.13 corresponds to one of the corridor trees described in Section 3.3. The first pursuer waits at the junction while

Problem	Edges	Nodes in G_c (Cells)	Nodes in G_I (Information)	Precomp. Time (sec)	Searching Time (sec)	Total Time
Fig. 3.8	28	25	200	0.04	0.02	0.06
Fig. 3.9	68	130	1727	0.44	0.12	0.56
Fig. 3.10	46	237	8787	0.53	1.59	2.12
Fig. 3.11	65	246	18830	0.87	9.86	10.73
Fig. 3.12	70	888	103049	3.00	168.63	171.63

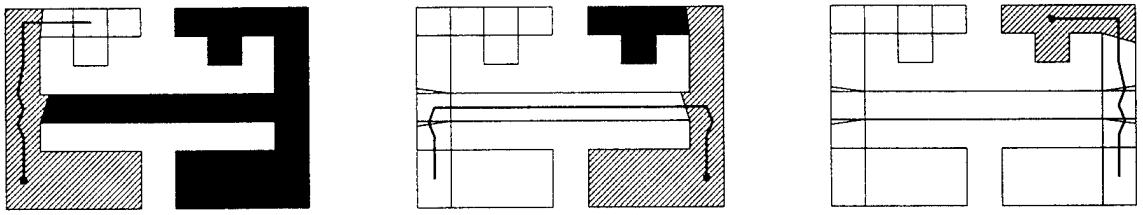
Figure 3.7: Various statistics are shown for the computed examples.

the second pursuer clears the remaining two components. Figure 3.14 requires two pursuers, and was solved by halting the first pursuer near the center hole (as shown in Figure 3.14.b) while the other pursuer cleared the remaining components. Figure 3.15 represents a very challenging example, which was solved using only two pursuers. We have run other examples which required up to four observers.

3.6 Discussion

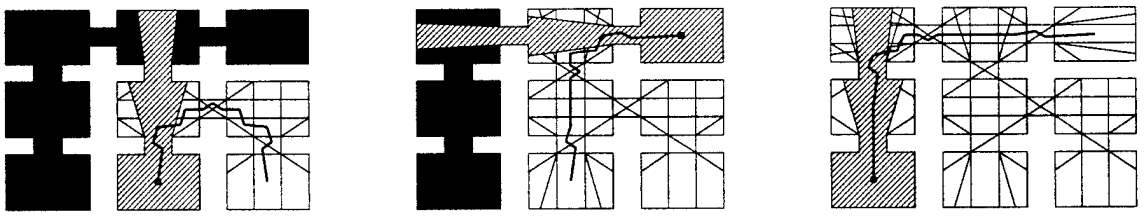
A visibility-based motion planning problem has been identified in this chapter that involves searching for an unpredictable target in a workspace that contains obstacles. Several bounds on the minimum number of needed pursuers were discussed. A general decomposition concept based on information *conservative* cells was introduced, which led to a efficient, complete algorithm for $H(F) = 1$ that has been implemented and tested. The algorithm was then augmented to coordinate multiple pursuers; this extension solves many problems, but does not generally yield the optimal number of pursuers.

Several variations and extensions of the problem are worth exploring. In addition to a visibility region, each pursuer can have a region of capture, and the task can be to capture the evader using one or more pursuers. Using the current evader model, only connectivity issues become critical for determining a solution strategy; however, the problem can be made more challenging by strengthening the model to include a bounded velocity, or possibly stochastic prediction. The topological issues could become significantly more complex for 3-D free spaces. Many vision systems have a limited field of view, and our problem can be adapted to planning strategies that sweep viewing angles in addition to moving the pursuers. Finally, a cost functional could be additionally defined, leading to problems such as finding the evader in minimum time.



a. b. c.

Figure 3.8: A computed solution trajectory is shown in three frames. The black area represents the contaminated region, and the white area represents the cleared region. The thick curve shows a portion of computed trajectory, which is continued in each frame. The shaded region indicates the visibility region at the final time step of the indicated portion of the trajectory. The thin lines in the cleared region indicate the cell boundaries. In the final snapshot, there is no place remaining where the evader could be hiding.



a. b. c.

Figure 3.9: Another computed example.

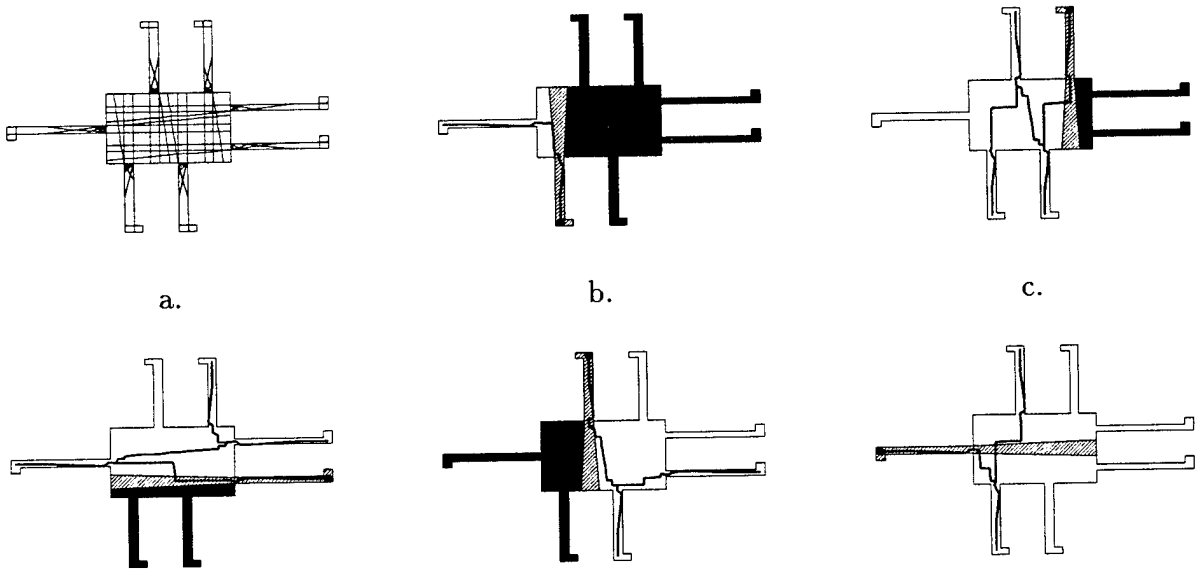


Figure 3.10: This difficult example requires two recontaminations of the leftmost corridor.

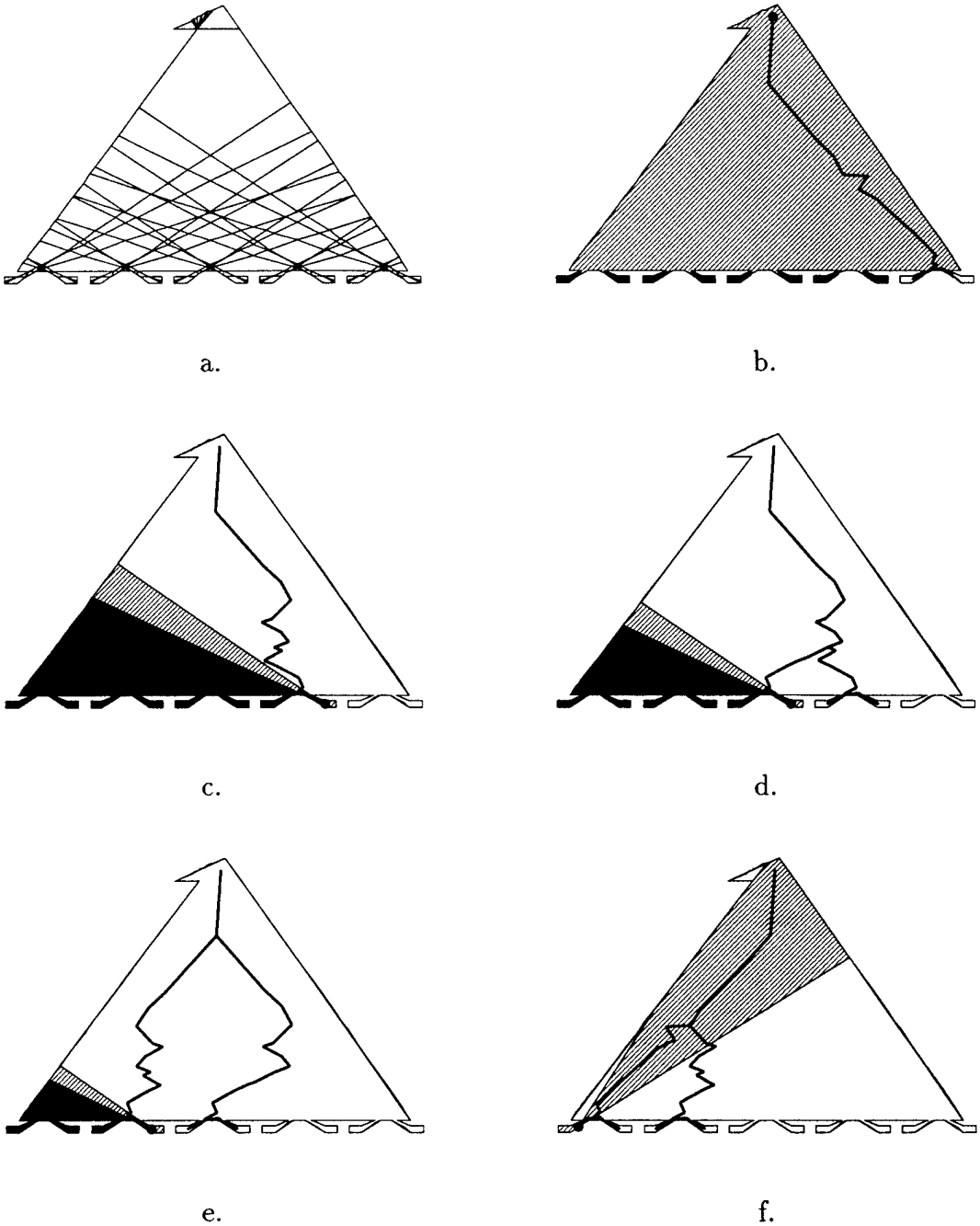


Figure 3.11: This example requires three recontaminations, and represents one in the sequence that requires a linear number of recontaminations.

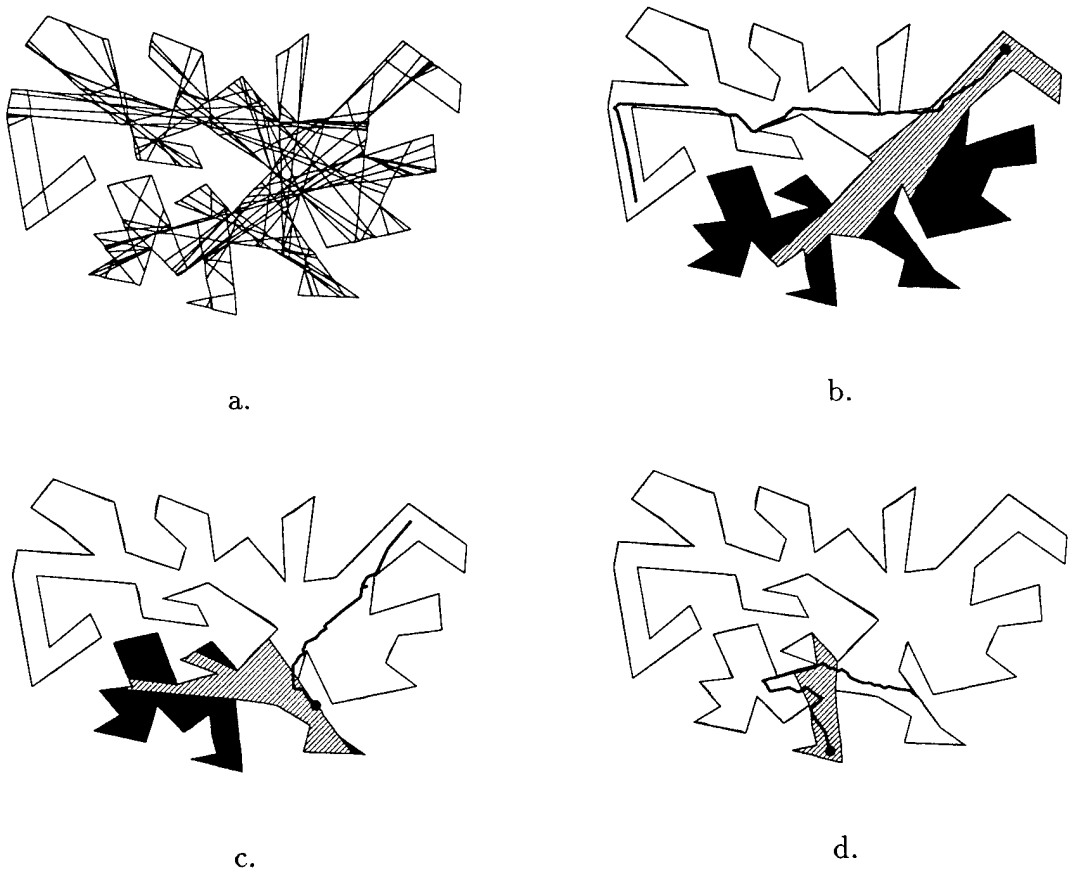


Figure 3.12: This bad example yields many edge-visibility cells.

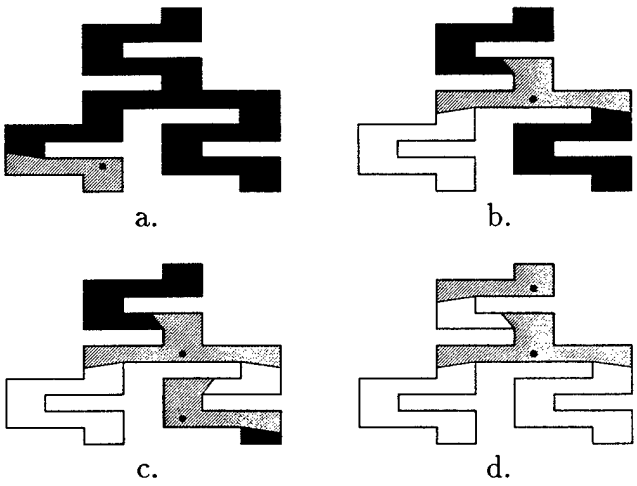


Figure 3.13: This simply-connected free space requires two pursuers. The first pursuer stops at the junction.

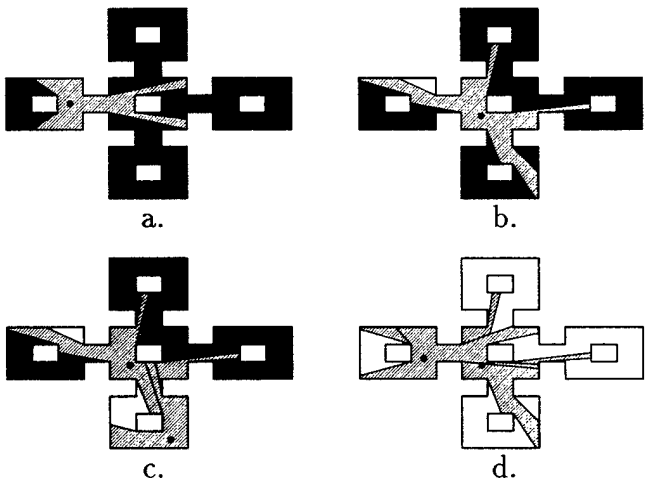


Figure 3.14: This example requires two pursuers.

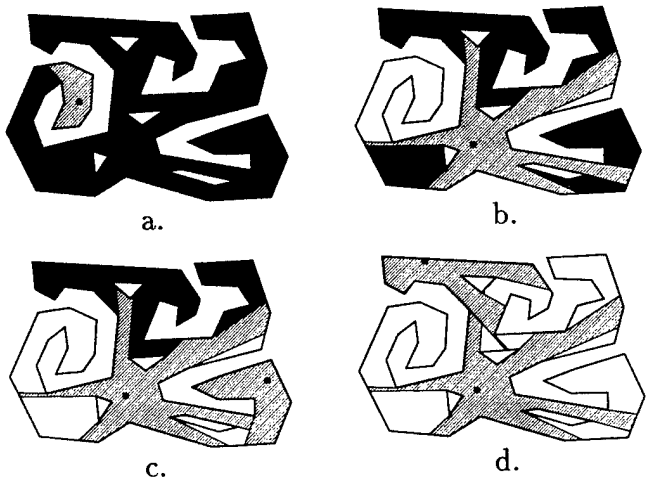


Figure 3.15: This complicated example was solved with only two pursuers.

Chapter 4

Target Tracking

This chapter presents our detailed progress on the problem of computing robot motion strategies that maintain visibility of a moving target in a cluttered workspace. A brief overview of this problem was given in Section 1.2.3. Both motion constraints (as considered in standard motion planning) and visibility constraints (as considered in visual tracking) must be satisfied. Additional criteria, such as the total distance traveled, can be optimized. The general problem is divided into two categories, on the basis of whether the target is predictable. For the predictable case, an algorithm that computes optimal, numerical solutions is presented. For the more challenging case of a partially-predictable target, two on-line algorithms are presented that each attempt to maintain future visibility with limited prediction. One strategy maximizes the probability that the target will remain in view in a subsequent time step, and the other maximizes the minimum time in which the target could escape the visibility region. We additionally discuss issues resulting from our implementation and experiments on a mobile robot system.

4.1 Introduction

Several applications require persistent monitoring of a moving target by a controllable vision system. In applications that involve automated processes that need to be monitored, such as in an assembly workcell, parts or subassemblies might need to be verified for accuracy or are determined to be in correct configurations. Visual monitoring tasks are also suitable for mobile robot applications [10]. In medical applications, one would like to move cameras around a surgery site to keep a designated area of interest (key tissue) in continuous sight, despite unpredictable motions of potentially obstructing people and instruments, and display a smooth sequence of images for the surgeon [48]. In a telepresence or virtual presence

application, a vision system can be used in a remote location to automatically track a variety of moving objects such as vehicles, people, or other robots. Visual information can also be used to track robots or robot features that appear in an image, and be directly integrated into a servo loop (e.g., [25, 35, 60]).

A motion planning problem is considered in this chapter in which a robot carries a camera that must maintain visibility of a target. The primary distinction between the problem considered in this chapter and standard tracking problems is the introduction of global, geometric constraints on both visibility and robot configurations. The following conditions are assumed: 1) an *observer* must maintain visibility of a moving *target*; 2) the workspace contains static obstacles that prohibit certain configurations of both the observer and target; 3) the workspace also contains static obstacles that occlude the target from the observer; 4) a (possibly partial) motion model is known for the target. The first condition implies that target tracking is the primary interest, and visibility can be defined in a variety of ways, depending on the particular problem. The second condition introduces the geometric constraints that appear in the standard path planning problem [46]. The third condition complicates the tracking problem by prohibiting *pairs* of observer and target configurations at which the observer cannot "see" the target. In many cases an obstacle in the workspace will cause both motion and visibility constraints. The fourth condition provides predictive information that should be utilized when designing a strategy. For example, the entire trajectory of the target might be known, or alternatively, only a velocity bound might be known. In addition to the previous four conditions, it may also be important to optimize some criteria such as the total distance traveled, energy utilized by the observer, or the quality of the visual information.

Section 4.2 provides a precise formulation of the problem in terms of configuration space concepts and system theory concepts. The computation methods presented in this chapter are divided into two sections on the basis of target predictability. Section 4.3 presents an off-line algorithm that determines optimal, numerical solutions for the case in which the target is completely predictable (i.e., the trajectory is known). The case in which the target is only partially-predictable is considerably more difficult, and is covered in Section 4.5. For this case, two different algorithms are presented that make on-line decisions that attempt to maintain future visibility. Both of these algorithms can be used in a real-time application using on-line information, and experimental results using two Nomad 200 robots are presented. Conclusions and discussion appear in Section 4.6.

4.2 Problem Formulation

Suppose that an *observer* and a *target* exist in a bounded, Euclidean workspace that is cluttered with static obstacles. In general, the observer and target can be considered as rigid or articulated bodies, with standard configuration-space parameterizations [46]. Let $\mathcal{C}^o_{free}$ and $\mathcal{C}^t_{free}$ denote the free configuration spaces of the observer and target, respectively. Let $X = \mathcal{C}^o_{free} \times \mathcal{C}^t_{free}$ represent the *state space*. A state simultaneously specifies configurations for the observer and target.

Motion models will next be formulated for the observer and the target. Discrete-time representations will be used to facilitate the expressions when there is uncertainty in target prediction; however, continuous-time representations could alternatively be used. Let the index k refer to the stage or time step that occurs at time $(k-1)\Delta t$, for some fixed, Δt (which specifies the sampling rate). Let $\mathbf{q}^o_k$ and $\mathbf{q}^t_k$ denote specific configurations at stage k, for the observer and target, respectively.

The observer will be controlled through actions, u_k, chosen from some action space U. The discrete-time trajectory will be given by a transition equation of the form $\mathbf{q}^o_{k+1} = f^o(\mathbf{q}^o_k, u_k)$, which yields a new configuration of the observer for a given current configuration and action. Constraints that include nonholonomy and bounded velocity can be modeled using f^o.

The target will be described by a similar transition equation; however, the actions that control the target are generally unknown to the observer. Let $\mathbf{q}^t_{k+1} = f^t(\mathbf{q}^t_k, \theta_k)$, in which θ_k represents unknown actions, chosen from some space Θ. One important special case, which is the subject of Section 4.3, is when the target is *predictable*. In this case the transition equation can be represented as $\mathbf{q}^t_{k+1} = f^t(\mathbf{q}^t_k)$.

Together, f^o and f^t define a state transition equation of the form $x_{k+1} = f(x_k, u_k, \theta_k)$.

Recall that each state, x_k, represents a pair of configurations, $\mathbf{q}^o_k$ and $\mathbf{q}^t_k$. A binary relation on these configuration pairs can be defined that declares whether the target is visible to the observer. This visibility can be defined in a number of ways. For example, the observer may have a 360-degree field of view and the target may be a point in the workspace. In this case, the target can be defined to be visible if the line-of-sight to the observer is unobstructed. In other examples, the field of view may be limited to some fixed cone, have a limited distance range, or the target may be polygonal and must be at least partially in view. Let $X_o \subset X$ represent the *visibility subspace*, which corresponds to the set of all states for which the visibility relation holds.

Next consider evaluating a state trajectory. Abstractly, the goal is to control the observer to ensure that the state remains in X_o. The loss or cost of applying control inputs and

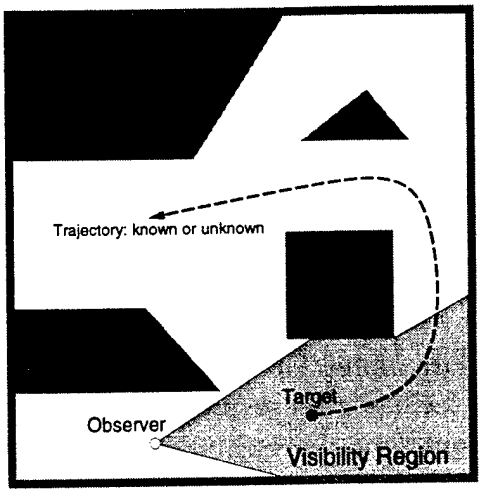


Figure 4.1: The goal is to maintain visibility of a moving target that may or may not be predictable.

obtaining a certain state trajectory can be generally expressed as

$$L(x_1, \ldots, x_{K+1}, u_1, \ldots, u_K) = \sum_{k=1}^{K} l_k(x_k, u_k) + L_{K+1}(x_{K+1}), \tag{4.1}$$

in which K represents the final time increment for issuing a action and $l_k(x_k, u_k)$ is a loss that accumulates in a single time step[1]. The final state, x_{K+1} can also be penalized, using L_{K+1}.

A simple, useful form of l_k is

$$l_k(x_k, u_k) = \begin{cases} 0 & \text{If } x_k \in X_o \\ \\ 1 & \text{Otherwise} \end{cases} . \tag{4.2}$$

This loss functional measures the amount of time that the target is not visible. One could also include a cost for choosing actions that produce motion. This would allow the robot motions to be optimized in addition to the time that the target is in view, and is considered in Section 4.3.

The loss functional evaluates a given trajectory. In the case of perfect target predictability, the trajectory can be inferred once the actions, $\{u_1, \ldots, u_K\}$ are specified to control the observer. In the case of a partially-predictable target, the loss functional is used to evaluate a state-feedback strategy using expected-case or worst-case analysis. These concepts are deferred until Section 4.5.

[1]An infinite number of stages could also be considered with discounted or average loss-per-stage.

4.3 Predictable Targets

In this section the assumption is made that $\mathbf{q}_k^t$ is known for all $k \in \{1, \ldots, K+1\}$. In this case, the state transition equation reduces to $x_{k+1} = f(x_k, u_k)$, which implies that the state trajectory, $\{x_2, \ldots, x_{K+1}\}$ is known once x_1 and inputs $\{u_1, \ldots, u_K\}$ are given.

For a problem that does not involve optimizing the robot trajectory, motions for the observer could be determined by recursively computing visibility and accessibility sets from stage $K+1$ down to stage 1. Suppose the target and observer are both points. Let V_k denote the subset of the free space from which the target is visible. Let A_{k_1} denote the set of all locations from which the observer could move at stage $k-1$ and lie in V_k at stage k. At any given stage, the observer must lie in $V_k \cap A_k$. A feasible trajectory for the observer can be obtained by backchaining from the final stage, guaranteeing accessibility and visibility in each step, until a set of possible initial states is obtained.

In the remainder of this section, a method that employs the dynamic programming principle to minimize a specific loss functional of the form (4.1) is presented. Although the approach shares some similarities with Dijkstra's algorithm on graphs, the principle is applied in the present context over a continuous state space and over discrete time. Its use follows directly from the differential equations that express the dynamic programming principle in standard optimal control [45, 47].

4.3.1 Computational Approach

The computational approach can be organized into four basic steps:

1. Construct a discretized representation of $\mathcal{C}_{free}^o \times \mathbf{K}$, in which $\mathbf{K} = \{1, \ldots, K+1\}$.
2. For each k, mark all discretized values of $\mathbf{q}_k^o$ from which the target (at known configuration $\mathbf{q}_k^t$) is visible.
3. Within X_o for each stage from $K+1$ to 1 perform dynamic programming computations with interpolation.
4. Extract the optimal sequence, $\{u_1^*, \ldots, u_K^*\}$, using *cost-to-go* representations.

Step 1: Because the target is predictable and parameterized through the stage index k, the subspace corresponding to $\mathcal{C}_{free}^o$ only needs to be considered as opposed to the entire state space. The stage index is included because the problem is time-varying. This is similar to the use of configuration-time space representations for motion planning among known moving

obstacles [40, 46]. An array representation of the spaces is constructed, which ultimately limits the current approach to observers that have only a few degrees of freedom. In [41], it is shown that the Fast Fourier Transform can be used to efficiently obtain a C-space representation from the static obstacles and robot geometry.

Step 2: Since the primary task is to maintain visibility of the target, the acceptable observer locations are marked. Using omnidirectional visibility, this can be accomplished efficiently by performing scan conversion of a computed visibility polygon that emanates from the target. Using a standard sweep algorithm [58], the visibility polygon can be computed in $O(n \lg n)$ time. If a loss functional is defined that evaluates individual viewpoints then one would precompute real-valued costs, as opposed to binary flags. For example, one might indicate a preference for a certain distance between the target and observer that is reflected in the real-valued costs.

Step 3: The dynamic programming computations are the most significant portion of the computation. For each stage k a *cost-to-go* function, $L_k^* : X \rightarrow \Re$, is computed using the cost-to-go function of stage $k+1$. The cost-to-go function represents the loss or cost that will be ultimately accumulated by starting from configuration $\mathbf{q}_k^o$ and choosing the optimal action at each stage. Due to the dependencies between the cost-to-go functions, L_{K+1}^* is computed initially, and a cost-to-go function is computed for each prior stage until L_1^* is finished. Although the domain of the cost-to-go function is X, computations only need to be performed on $\mathcal{C}_{free}^o$ because the target configuration is fixed for each stage.

The dynamic programming principle defines the relationship between the cost-to-go functions:

$$L_k^*(x_k) = \min_{u_k} \{ l_k(x_k, u_k) + L_k^*(x_{k+1}) \}, \tag{4.3}$$

in which l_k is defined in the loss functional (4.1), and x_{k+1} is obtained for each choice of u_k through the state transition equation, $x_{k+1} = f(x_k, u_k)$. The difference equation (4.3) defines a very local relationship between successive cost-to-go functions, yet the principle of optimality ensures that a globally optimal strategy will result.

One difficulty results from using discretized representations of the continuous dynamic programming principle. The next state, x_{k+1}, will usually not lie exactly at a discretized value. Rather than simply looking up L_{k_1} at the nearest quantized value, the value of L_{k+1} can be computed through linear interpolation of the cost-to-go values between all neighbors of x_{k+1}. This technique has been used previously in numerical dynamic programming computations; related issues are discussed in [45].

Step 4: Suppose the cost-to-go functions have been computed. If the $\mathbf{q}_1^o$ is fixed, then u_1^* can be obtained by using (4.3) for the given initial state, x_1. If $\mathbf{q}_1^o$ is free, then an optimal initial configuration can be obtained by selecting the configuration $\mathbf{q}_1^o$ minimizes L_1^*. Once u_1^* has been determined, the next state $x_2 = f(x_1, u_1^*)$ can be inferred. Equation (4.3) can

be used again to determine u_2^*. This process iterates until the final, optimal action u_K^* is obtained.

The time complexity of the method is linear in the number of actions and in the number of stages, but is exponential in the dimension of the observer configuration space. On a SPARC 20 workstation the dynamic programming computation time varies from about twenty seconds to several minutes, depending on the difficulty of the example and the chosen resolutions.

4.3.2 Computed Examples

The optimal strategies were computed for several strategies, and the resulting observer trajectories were simulated. Some of these results are shown in this section to demonstrate the method and to illustrate the global aspects of this motion planning problem. In all of the examples, the workspace is 2-D with dimensions 100 units by 100 units. All obstacles obstruct both visibility and motion. The target moves at its maximum speed of 3 units/stage. The observer is controlled through the simple holonomic model:

$$\mathbf{q}_{k+1}^o = \mathbf{q}_k^o + u_k^1 \Delta t \begin{bmatrix} cos(u_k^2) \\ sin(u_k^2) \end{bmatrix}, \tag{4.4}$$

in which $u_k^1 \geq 0$ represents the speed of the observer, and $u_k^2 \in [0, 2\pi)$ represents the direction of motion. No dynamics are considered in this model.

Figure 4.2 shows a simulation of a trajectory that is obtained from the computed optimal strategy. The actions $u_k^1 = 0$ (no motion) and $u_k^1 = 3$ and $u_k^2 \in [0, 2\pi)$ (motion at a fixed speed in some direction) were allowed. The loss functional is of the form $l_k(x_k, u_k) = l_m + l_v$. The term l_m is a penalty for motion, and $l_m = 0$ if $u_k^1 = 0$, otherwise $l_m = 1$. The term l_v is a penalty for losing visibility, which is much more important, yielding $l_v = 500$. There are 105 stages for this problem, and the dynamic programming computations took about 20 seconds on a SPARC 20 workstation. Note that although the target trajectory is quite long, the distance traveled by the observer is short. An initial position for the observer was automatically selected from which the target was visible during the first portion of the trajectory. The observer moves just barely far enough to the lower left, and then finishes by remaining in the lower right for the final segment of the target trajectory.

The example in Figure 4.3 involves the same geometry; however, the loss functional is slightly changed to yield $l_v = 0$ only if the target is both visible and the distance between the target and observer lies within 10 and 25 units. Also, the speed, u_k^1, is allowed to vary between 0 and 3, with a loss, l_m, that proportional to the speed. In this case, the observer must travel a greater total distance, yet an optimized trajectory that maintains visibility is still obtained.

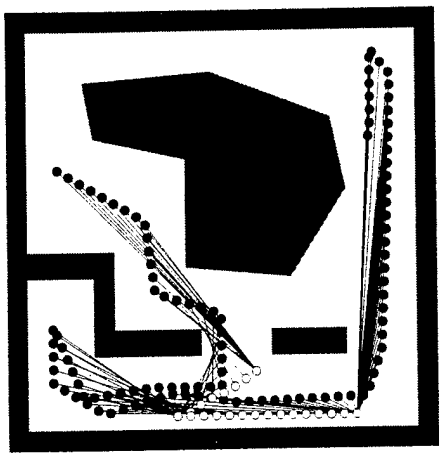


Figure 4.2: A simulation is shown of the optimal state trajectory for a tracking problem. The observer strategy, including the initial position, is chosen to maintain visibility and minimize the total distance traveled. The target positions are shown as black circles, and the target trajectory starts in the upper left. The observer positions are shown with white circles. The line-of-sight is shown between the observer and target at each stage.

Figure 4.4 shows another example that requires the distance between the target and observer to remain within 10 and 25 units. The intial position of the observer is initially specified for this problem, and both the observer and target start in the upper right.

Figure 4.5 shows an example in which the maximum observer speed is 1.5, and the target speed is 3.0. There are many visual obstructions, and the observer is able to maintain visibility of the target in all but two stages. During this period, the observer moves quickly to the right to reacquire the target.

Finally, Figure 4.6 shows an example that illustrates how the optimized observer trajectory can be lengthened due to the visibility constraint. In the example, the target sharply turns into a corridor, and the observer is forced to swing out further into the open area to maintain visibility.

4.4 Nearly-Predictably Targets

Consider a problem in which there are a finite number of goal regions in the environment that the target periodically visits. Assume that once a goal has been assigned, the target trajectory will be completely predictable. The uncertainty, however, comes about because

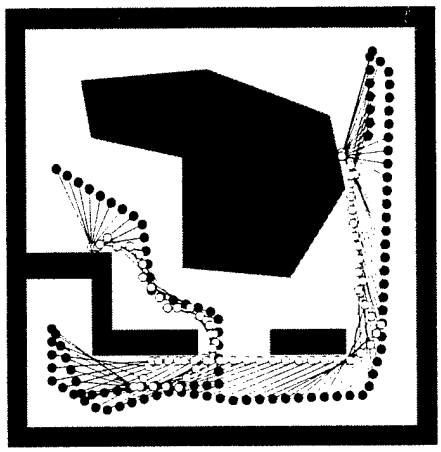


Figure 4.3: This example differs from the previous one by additionally requiring that the observer remains within a specified distance range from the target.

the observers do not know which goal has been assigned. In this case, the observer knows that the target will traverse one of a finite collection of paths, and attempts to maintain visibility under all possible choices. For this problem we have designed and implemented an algorithm that moves multiple observers to maintain visibility of the target. Each observer can vary its motions during execution in response to the actual target trajectory.

The target is restricted to move along a predefined network of paths, but its decision at each node of this network is unpredictable. We have developed a planner which generates optimal motion strategy for solutions which require a single observer. The planner is based on dynamic programming approach by backchaining solutions from all possible target goal configurations to its initial configuration.

First, the planner divides the workspace into a grid and computes the set of visible grid positions to the target at each point along the path. Next, the planner recursively computes a set of grid positions for the observers to maintain visibility with the target from its current position to all possible final configurations. This set of grid positions is called the *trackable set.* The trackable set is computed by backward chaining from the final target position's trackable set. The trackable set of the target's final configuration is the visible set. At each time unit, the trackable set is enlarged to account for the observer's maximal movement speed. The trackable set from a previous time step is computed by finding the intersection of target's visible set and the enlarged trackable set at the next discretized step. At the merging of the paths, the trackable set is just the intersection of all enlarged trackable sets from the next time step with the current visible set.

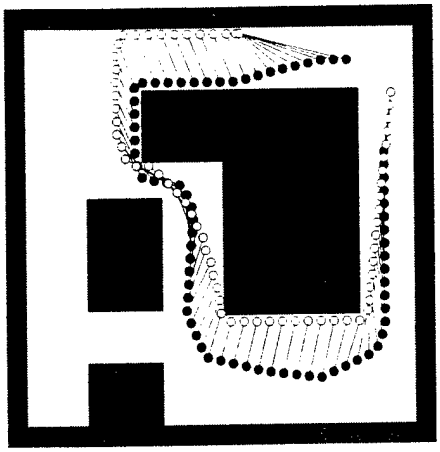


Figure 4.4: This example shows an optimized observer trajectory that maintains a specified distance range, and the initial observer position is fixed.

This planner will find a motion strategy for one observer if a solution exists. The planner is resolution complete for problems that require only one observer. However, for problems requiring more than one observer, the planner uses greedy heuristics to make local decisions which may turn out to be suboptimal in the total number of observers used.

4.5 Partially-Predictable Targets

The problem in Section 4.3 allowed restricting the state space to the observer configuration space because of target predictability. In this section it is assumed that only weak information, such as a velocity bound, is known regarding the target. In principle, a dynamic programming approach can be taken to determine optimal strategies for the partially-predictable case; however, even for a simple planar problem the state space is four-dimensional. This increased complexity motivates the consideration of alternative approaches which can provide reasonable behavior by making a tradeoff between computational cost and the quality of the solution. Experiments with an on-line algorithm that is presented in this section were performed using a mobile robot system that is described in [6].

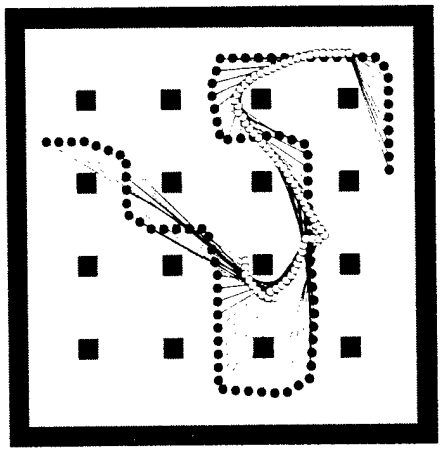


Figure 4.5: In this example, the target can moves at twice the maximum speed of the observer. In the optimal trajectory of the observer, there are only two stages in which the target is not visible.

4.5.1 Optimal Strategies

The notions of a strategy and of optimality become more interesting if there is uncertainty in target prediction. Recall that $\theta_k \in \Theta$ refers to an unknown action that can be applied to move the target using $\mathbf{q}_{k+1}^t = f^t(\mathbf{q}_k^t, \theta_k)$. Two alternative interpretations of the unknown actions are possible. If the unknown actions are modeled as *nondeterministic uncertainty* (as referred to in [23, 47]), then it is only assumed that $\theta_k \in \Theta$ for some specified Θ. In this case, one would design a strategy that performs the best given the worst-case choices for θ_k. Alternatively, a *probabilistic uncertainty* model can be used, in which it is additionally assumed that $p(\theta_k)$ is given, in which $p(\cdot)$ denotes a probability density function. In this case, one could design a strategy that minimizes the loss in the *expected* sense.

Because the state trajectory cannot be predicted, a state-feedback strategy is designed, as opposed to directly specifying the actions (the actions must respond to on-line changes). This represents a standard notion used in optimal control, and used been applied to motion planning problems that involve other forms of prediction uncertainty in [47]. Let $\gamma_k : X \rightarrow U$ denote a *strategy at stage k*. Let $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_K\}$ denote a *strategy*. Let Γ denote the space of possible strategies for the robot.

For the case of nondeterministic uncertainty, a strategy, $\gamma^* \in \Gamma$, can be selected that yields the smallest worst-case loss:

$$\check{L}(x_1, \gamma^*) = \inf_{\gamma \in \Gamma} \check{L}(x_1, \gamma) = \inf_{\gamma \in \Gamma} \sup_{\gamma^\theta \in \Gamma^\theta} L(x_1, \gamma, \gamma^\theta) \tag{4.5}$$

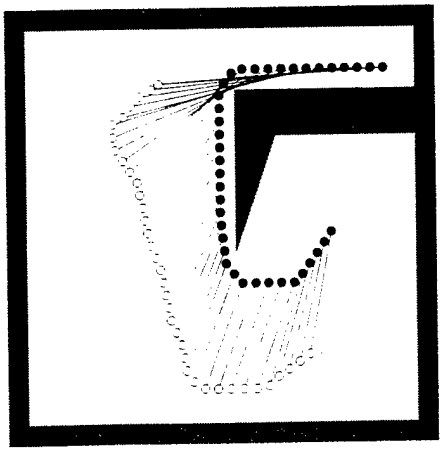


Figure 4.6: This example shows how the observer must swing outward, ultimately following a longer path, to maintain visibility of the target. The initial observer position was fixed.

for all $x_1 \in X$, and γ^θ represents a choice of θ_k for every stage. This indicates that from any initial state, the strategy will guarantee the least possible loss given the worst-case actions of nature. This concept has been used previously to design controllers based on worst-case analysis [1].

With probabilistic uncertainty, a strategy, $\gamma^* \in \Gamma$, can be chosen that minimizes the expected loss:

$$\bar{L}(x_1, \gamma^*) = \inf_{\gamma \in \Gamma} \bar{L}(x_1, \gamma) = \inf_{\gamma \in \Gamma} \int L(x_1, \gamma, \theta) p(\theta) d\theta \tag{4.6}$$

for all $x_1 \in X$. This corresponds to selecting a strategy that minimizes the loss in the expected sense, as considered in stochastic optimal control theory [7, 44].

These expressions capture the general design problem; however, for most practical problems, it may be preferable to consider approximate or simplified strategies. Sections 4.5.2 and 4.5.3 describe two on-line approaches that attempt to optimize local criteria that are related to the global task.

4.5.2 Maximizing the Probability of Future Visibility

For a given current state, x_k, this approach selects an action u_k that will maximize the probability that the target will remain in view at stage $k+1$. This approach can be viewed as a tradeoff that involves limited consideration of future states. An action could alternatively be chosen that maximizes the probability that the target will remain visible over the next

m states; however, the computational cost could increase dramatically.

A probabilistic uncertainty model is used for target prediction. Hence, it is assumed that $p(\theta)$ is given, from which a density $p(\mathbf{q}^t_{k+1}|x_k)$ can be obtained using the motion model for the target. Formally, $u^*_k \in U$ is selected to maximize

$$P[\mathbf{q}^t_{k+1} \in V(\mathbf{q}^o_{k+1})] = \int_{X_o} p(x_{k+1}|x_k, u_k), \tag{4.7}$$

in which $V(\mathbf{q}^o)$ represents the set of target configurations at which the the target is visible to the observer at configuration $\mathbf{q}^o$. This defines a state-feedback strategy $\gamma(x_k) = u_k$.

Assume that both the observer and the target travel in a planar workspace, and that (4.4) represents the observer motion model. The observer has an additional degree of freedom that corresponds to the direction that a mounted camera is aimed. Assume that little is known about the target other than its maximum speed, $\|v^t\|$, and that $p(\mathbf{q}^t_{k+1}|x_k)$ is a disk of uniform probability density with radius $\|v^t\|$, centered at $\mathbf{q}^t_k$. Zero probability mass is obtained, however, in regions within the disk that correspond to configuration-space obstacles. The causes geometric information to be utilized.

Because the approach involves a significant tradeoff from optimality, experimental studies were performed to assess its utility. We have performed numerous experiments using two Nomad 200 mobile robots, one of which is equipped with a vision system (see Figure 4.7). A visibility region $V(\mathbf{q}^o)$ is defined as a cone with apex at the observer and truncated at a minimum and maximum distance.

The planner is integrated into a larger system composed of several modules engaged in landmark-based localization, motion and camera control, and image feature tracking. The current implementation of the system consists of eight modules: a visual target tracker, a landmark detector, the motion planner, a graphic interface, a calibration module, the control module, and a low-level control driver. The visual target tracker consists of a simple detection algorithm that identifies a pattern placed on top of the robot serving as the target, and reports new relative locations of the target every 0.3 secs. The landmark detector samples the ceiling every second and reports relative positions, orientation, and identification of any landmark within the visual range of the camera. The motion planner receives information concerning the target and observer configuration in order to compute the observer new position. The control module receives information from the robot encoders, the target tracker, and the landmark detector to estimate the configuration of the whole system and guides the robot to the positions specified by the planner. The calibration modules computes the physical configuration of the cameras onboard the observer. Finally, the graphic interface allows the user to interactively supervise and control the experiment.

We have also studied the observer behavior in a variety of simulated environments. A sequence of frames from one such simulation is shown in Figure 4.8.

Figure 4.7: Two Nomad 200 robots are used for the experiments. One is equipped with a camera that is mounted on a rotary base.

Performance can generally be improved by incorporating additional information into the prediction. For example, the current heading of the target can be utilized to provide more realistic expressions of $p(\mathbf{q}_{k+1}^t|x_k)$. A dynamic model of the target could be utilized to improve prediction for on-line strategies.

For nonreactive targets the motion prediction problem is completely independent from the motion planning one. The target future position distribution may be computed numerically and in some cases analytically. We have studied both approaches for the case of a *Cartesian* target, in which independent actuators drive the target along different orthogonal motions. We assume that the target decides a new acceleration action every time τ, which will be the *target's sampling rate* (not to be confused with Δt, the *planner's scope*). The values of acceleration will be selected from a set $\mathcal{A}$, which is the *acceleration possibility set.* We presuppose that the acceleration values at each step are i.i.d. random variables.

Typically the planner scope Δt is larger than the target sampling rate τ, hence we are interested in predicting the probable target locations after $n = \Delta t/\tau$ steps given a measurement of the target configuration at stage k. The prediction is then used to compute the observer location that maximizes the probability of observation at stage $k+1$. If n is small and $\mathcal{A}$ is a finite and countable set then the distribution may be computed numerically by evaluating the possible sequences of actions, computing the resultant final configurations, and storing the associated probabilities in a data structure.

If the size of $\mathcal{A}$ is m (the number of alternatives) then after n steps there are m^n possible target configurations, so the search space is exponential. However, the discretized equations

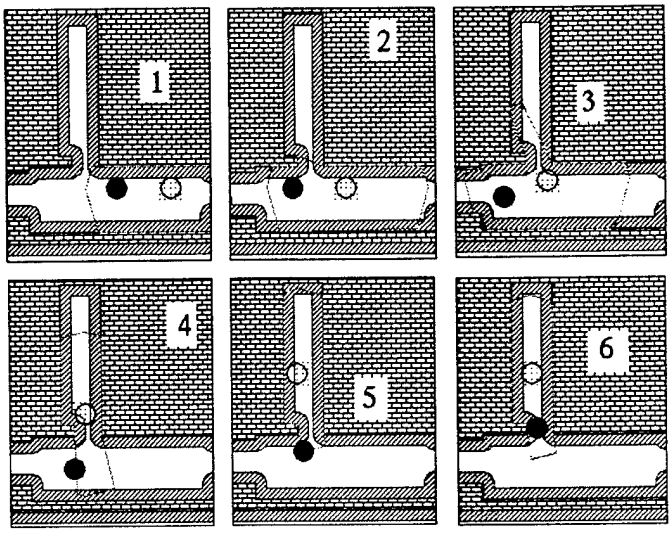


Figure 4.8: A sample execution is shown of the on-line planner. The black disk is the observer.

of motion can be formulated in such way that most of the operations can be precomputed. Execution can also be improved by detecting equivalent sequences of actions in order to transform the search tree into a lattice.

For n large enough ($\gg 30$) the resultant distribution can be proved to be Gaussian. If the possible accelerations are i.i.d. with mean μ and variance σ^2, then the location of the cartesian target after n steps given an initial position and velocity x_0 and v_0 is described by $x_n = x_0 + n\tau v_0 + S_v + S_a$, with S_v being a Gaussian distribution of mean $\frac{1}{2}\Delta t(\Delta t - \tau)\mu$ and variance $\frac{1}{6}\Delta t\tau(\Delta t - \tau)(2\Delta t - \tau)\sigma^2$, and S_a a Gaussian with mean $\frac{1}{2}\Delta t\tau\mu$ and variance $\frac{1}{4}\Delta t\tau^3\sigma^2$. Currently our efforts concerning target modeling are aimed toward the study of the effect of how obstacles distort the probability distribution of the target motions under this model.

4.5.3 Maximizing the Minimum Time to Escape

This section describes an alternative on-line approach that uses nondeterministic uncertainty and worst-case analysis as opposed to probabilistic analysis. Assume that the target has a maximum speed, $\|v^t\|$. Let $d(x_k)$, denote the distance from $\mathbf{q}^t$ to the nearest boundary of $V(\mathbf{q}^o)$. The *minimum time to escape*, t_{esc}, represents the smallest interval of time within which the target could escape if the observer remains at $\mathbf{q}^o$. Clearly, $t_{esc} \leq \|v^t\| d(x_k)$.

Assume that $V(\mathbf{q}^o)$ is a truncated cone, as in Section 4.5.2. The on-line strategy is to select

an action $u_k = \gamma_k(x_k)$ that maximizes t_{esc}. This corresponds to preparing for the worst-case motions of the target. Typically, $\Delta t \ll t_{esc}$, which implies that the strategy effectively considers target actions that are several stages into the future. This strategy appears to be an improvement over maximizing the probability of future visibility; however, it is still not a globally optimal solution. We are currently in the process of experimenting with this strategy to evaluate its performance.

4.6 Discussion

A research problem that involves maintaining visibility of a moving target has been identified and formally characterized. Workspace geometry introduces standard motion planning difficulties into the visual tracking problem. For the case of predictable targets, an algorithm was presented that provides numerical, optimal solutions for problems that have a low-dimensional observer configuration space. For the case of partially-predictable targets, optimal strategies were characterized, and two on-line strategies were presented. The strategy that maximizes the probability of future visibility has been tested in experimentation and simulation, and the strategy that maximizes the minimum time to escape is currently being evaluated.

Several interesting extensions and variations can be considered in future investigations. In future investigations it may become useful to consider approximation algorithms that can provide a solution that is within some bound of optimal. For example, one could randomly select and connect points in the freespace for high-dimensional problems with the possibility of stating performance bounds probabilistically, as in [5]. Many interesting possibilities exist for coordinating multiple observers and/or multiple targets. For example, several observers may be able to track a faster-moving target by making decisions to "cover" disparate regions of the cluttered environment. Another issue that could be addressed is imperfect information regarding the current configuration of the target. In practice, the target information is derived from image data, which loses information due to projections, noise, and quantization. It has been assumed in this chapter that the target motions do not depend on observer configurations. One could alternatively consider a *reactive* target that attempts to avoid being observed. Finally, the intentions of the target could be speculated (probabilistically) by the observer. For example, the target might make deliveries to one of several locations periodically. The observer can infer the route once the intention is known (or narrowed to few possibilities), and a significantly improved strategy can be computed.

Chapter 5

Implementation of an Autonomous Observer Prototype

5.1 Introduction

This chapter describes the implementation of an autonomous observer prototype. Our goal was to develop a system that provides a human user with intuitive, high-level control over a mobile robot which autonomously plans and executes motions to visually track a moving target (see Figure 5.1.a). The user receives real-time feedback, such as a graphical display of the positions of the observer and target overlaid on a map of the environment.

The system implementation brings together concepts and algorithms from computer vision, motion planning, and computer graphics in order to create a robust, useful, and integrated system. One important aspect of the system is that vision, often viewed merely as a mechanism for aiding robot navigation, itself becomes the central objective of the system.

The chapter is organized as follows. In Section 5.2 we present the overall design of the system and the roles of its different components. In Section 5.3 we describe an implementation of each component of the autonomous observer prototype. In Section 5.4 we describe the system's central module (main system controller). In Section 5.5 we describe our initial experiments with the autonomous observer. Finally, in Section 5.6 we draw conclusions from our experiments and discuss possible directions for new implementations.

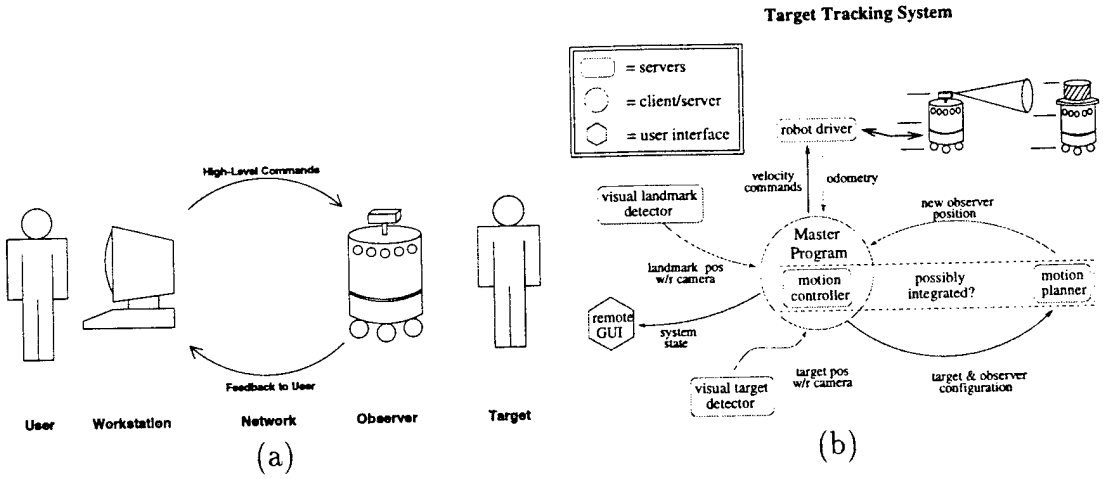


Figure 5.1: (a) Interaction between a user and the intelligent observer; (b) The components of the IO system.

5.2 Overall Design

The complete system consists of five major modules: landmark detection, target tracking, motion planning, user interface, and motion control. The relationships between these modules and the actual robot are shown in Figure 5.1.b. Each module is briefly described below.

5.2.1 Landmark Detection

As the observer moves around in its environment it must keep track of its current position. Our approach to this problem involves placing artificial landmarks throughout the environment. Many researchers have studied the use of landmarks in robot navigation (for examples see [34, 43, 49, 51]). Landmark detection is explained in detail in Appendix B.

In our system, the positions of the landmarks are provided as part of a map given to the observer. Each landmark induces a *landmark region*, within which the landmark is visible to the robot. The robot localizes itself by visually detecting landmarks and determining its position relative to them. Since the success of the robot depends on this self-localization, the vision algorithms used to detect the landmarks must be fast, accurate, and robust.

5.2.2 Pattern-tracking in an Image Sequence

The central task of the autonomous observer is to observe moving objects (or targets). There are two main requirements: first, that the observer recognizes when a new target enters its field of view; and second, that the observer is capable of tracking the desired target as it moves. Since the robot must respond to the movement of objects, all tracking must happen in real time. In general, target motions may be almost totally unconstrained, and the targets themselves may be non-rigid and of unknown shape. In order to handle such targets, we can apply real-time versions of tracking algorithms such as those described in [37]. Currently, however, we use a simplified approach as detailed in Section 5.3. Pattern-tracking in image sequences is developed at greater length in Appendix C.

5.2.3 Motion Planning

The autonomous observer goal is to keep in view a moving target. Since the target motion is not known in advance, we must employ an on-line algorithm to perform tracking. In order to maintain a view of the target we must avoid occlusions due to obstacles and, as in the traditional motion planning, we must also avoid collisions with them. Any given obstacle may obstruct the robot's view, or its motion, or both.

The implemented tacking algorithm is an on-line scheme based in maximizing the probability of future visibility. This approach is one of the on-line schemes proposed and described in detail in Chapter 4.5.2.

5.2.4 User Interface

One simple way of providing feedback to the user is to display live video from the robot's camera. There are, however, several drawbacks to this approach. First, it makes no use of the higher-level representation kept by the autonomous observer. Second, it limits the viewpoint to that of the robot's camera. Third, we would like to avoid "information overload" in cases where there are multiple cameras and/or observers. Finally, full-motion video requires high transmission bandwidth; this problem is especially important when the user is far away from the observer.

Instead, the user is presented with a synthetic reconstruction of the environment in which the observer is operating. This reconstruction can be either a two-dimensional overhead view or a three-dimensional view from an arbitrary vantage point. This module relies on an appropriate geometric model of the environment as well as information about the positions of the observer and the target.

Assuming that the environment is largely static, only a small amount of new information is required to update a scene, alleviating the bandwidth problem. This also allows us to fuse input from multiple cameras into a single, unified view.

5.2.5 Motion Controller

The motion controller takes the role of the top-level supervisor in the autonomous observer system (see Figure 5.1.b). It coordinates communication with the other components and produces appropriate low-level commands to control the physical robot. In addition, it periodically updates the current estimate of the robot's position based on feedback from the landmark detector and the odometric information from the robot. When appropriate, it also requests a new goal point from the motion planner. Finally, is sends updated information to the user interface.

The whole system is composed of a number of different processes, each of which has its own characteristic cycle time. Because of this, the motion controller must communicate asynchronously with the other processes, using new information as it becomes available. The exception is communication with the path planner, which follows a transaction-based model: the controller requests new a goal once it has achieved the previous one. The Motion Controller is presented in detail later in this chapter.

5.3 Implementation

Each component has been implemented as a separate Unix process. The communication between processes uses standard TCP/IP protocols, making it possible to run them on different machines to increase performance. In fact, during our experiments we ran the landmark detector, the target and the motion controller on a Pentium 90 computer on-board the robot. The motion planner runs on a Sun SPARCstation 20, while the user interface can runs on a number of different machines. The implementation of each process is detailed below.

5.3.1 Landmark Detection

As discussed earlier, we rely on artifical landmarks to localize the robot. Our landmarks, shown in Figure 5.7.a, are placed on the ceiling at known positions throughout the robot's workspace. Each landmark consists of a black square with a 4×4 pattern of smaller squares inside of it. The detection algorithm first identifies edge pixels in the image (using a variant

of the algorithm presented in [15]), and then looks for edge chains that are consistent with the boundary of a square. When such a chain is found, the corners are detected and then lines are fitted to the pixels which make up each edge. The slopes of these lines yield the orientation of the landmark; their intersection points locate the landmark's corners. Once the landmark is localized, the positions of the inner squares are computed and their intensities are read from the image using bilinear interpolation. These intensities are grouped into "black" and "white" subgroups to determine the 16 binary values they represent. Four of these values are used to disambiguate the landmark's orientation, and the others encode the landmark's unique ID.

The landmark detector uses 256×243 grayscale images as input. The robot does not need to relocalize at a high frequency rate –it is enough to correct every few seconds or so. In order to save computational resources on-board the robot, landmark localization is done at a rate of 0.2 seconds. The algorithm is very accurate: the translational error has zero mean and a standard deviation of 0.75 inches, while the rotational error has also zero mean and has a standard deviation of 0.5 degrees.

5.3.2 Target tracking

This component is used by the observer to detect and track a moving target. Currently the target consists of a number of black, vertical bars on a white cylindrical "hat" which sits on top of the target robot (see Figure 5.7.c). The tracking algorithm detects these bars in each image and, given the camera parameters and the physical size of the bars, computes the target's location relative to the camera.

The tracking system operates at approximately 5 frames per second using 256×243 grayscale images as input. For each frame, it determines the angle θ and distance r to the center of the target. Typically the target can be detected at distances ranging from 2 feet to 9 feet from the camera. Experimentation shows that θ is accurate to within ± 1 degree, and r is accurate to within ± 4 inches. However, changes in the image size of just one pixel can lead to changes of almost an inch when the target is far away. This noise may constitute a problem when the enire system executes, so some noise filtering is appropiate.

5.3.3 Motion planning

Our implemented solution to the on-line view planning problem is deliberately simple so that planning time is reasonable. The planner is given a polygonal map of the workspace, as well as bounds v_R and v_T on the maximum velocities of the observer and target, respectively. These velocities are given in terms of distance per planning cycle.

The workspace is represented by a 117×59 grid, with a resolution of about 6 inches per unit. Obstacles are represented by polygons. Many of the steps required to compute a motion plan are precomputed when this module is first invoked. Planning time takes less than 1 msec in a Sun SPARCstation 20, with about 66 seconds of precomputation.

5.3.4 User interface

In our initial system we have adopted a simple, two-dimensional user interface as shown in Figure 5.7.b. Physical obstacles are shown in black and configuration space obstacles are dark gray. The positions of the observer (R) and target (T) are updated in real time. In addition, the visibility region of the observer is shown in light gray.

The entire system can be executed from the World Wide Web. For detail go to the autonomous observer status page:
`http://robotics.Stanford.EDU/users/io`
and the execution page:
`http://robotics.Stanford.EDU/~dlin`.

5.3.5 Motion control

As mentioned before, this module not only controls the motion of the robot, but it also coordinates communication between all of the other modules. In a sense it is the top-level supervisory process for the whole system. It also attempts to compensate the lack of an omnidirectional camera for target tracking by controlling the turret rotation so as to keep the observer's camera pointed toward the pattern on the target.

The motion controller is explained in detail in the next section.

5.4 Motion Controller

The motion controller is the central module in the target tracking subsystem of the autonomous observer. It serves as the communication hub of the other modules and generates velocity commands to move the robot according to the goals specified by the planner. As shown in Figure 5.2, the controller connects as a client to the vision and motion planning modules, while acting as a server for the remote user interface.

The networking role played by the motion controller is incidental to one of its key purposes, which is the fusion of the information generated by the distinct programs into a consistent

estimation of the system state. The motion controller computes and updates estimates of the observer and target positions with the information ensuing from landmark observations, odometry, and observations of the target. State information is sent to the motion planner (which requires it to compute new goal configurations for the observer), and to any external client which may be connected to the system. Since of all system modules the motion controller is the one which integrates all the information into a consistent view of the state, it is also natural that this module acts as the external communication interface to any possible clients (such as the graphical interface program).

The second principal task of the main controller is to constantly issue velocity commands to the three robot actuators (translation, steering, and turret rotation), in order to move the robot smoothly according to the evolution of the system state and the goals generated by the motion planner. Translation and steering are actuated in order to satisfy the goals given by the planner, while the turret is controlled so as to keep the camera oriented to the target by using the readings from the visual tracking module as feedback.

The main challenge in designing the control laws for the actuator motions is the presence of communication delays between the system components. Due to the asynchronous nature of networking, these delays are *not* periodic, a characteristic which complicates feedback design.

In this appendix we present the formulation of the kinematic and target error models, which leads to the design of a set of control laws that can be easily tuned based on knowledge of the system architecture, thus making "trial and error" adjustments unnecessary or minimum. This characteristic is very useful in a modular system, where each part may be enhanced independently of the rest.

5.4.1 Description of the Kinematic and Target Error Models

The kinematic and target error models are key elements of a good tracking system. The kinematic model allows us to efficiently operate the robot with velocity commands taking into consideration delays in the network, information from landmarks and encoders, and the kinematic limitations of the robot. The target error model is useful to reconstruct the position of the target given the information provided by the vision module; information which, due to the inherent complexity of any vision system, is already outdated when a frame processing is completed.

The kinematic error model is relevant to the controller, while the target error model is relevant to the controller *and* to any other module which uses the target position in its calculations (such as the case of the planner module). Hopefully the models described here will prove useful in other implementations of the IO besides our own, especially the kinematic model, which might lead to the design of more efficient control laws.

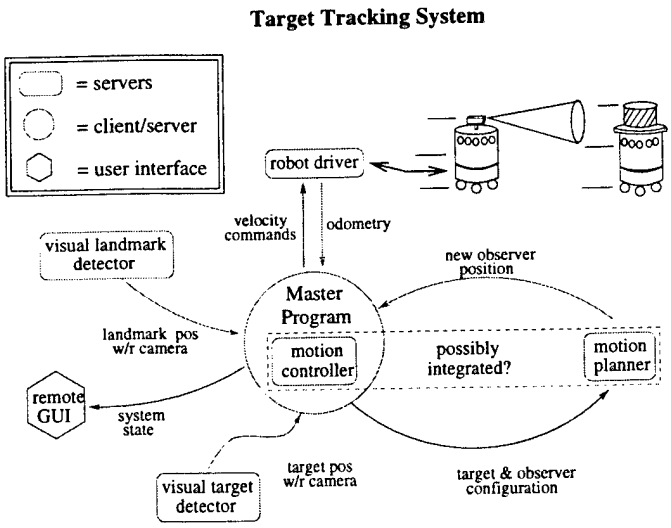


Figure 5.2: Architecture of the Target-Tracking System

The Kinematic Model

Communication between modules is asynchronous, and there is no guaranteed bandwidth for any specific socket in the system. The time lag found in the system's closed loop is in general not constant, as it depends on the network load. Since the communication between modules is asynchronous, communication delays cannot be neglected. Figure 5.3 shows a reasonable model of how the sequence of commands and state information requests is carried out.

The robot itself has a low-level motion control program running in a DMC-630 board. The DMC-630 sets the robot's axis positions and/or speeds with a set of speeds and/or accelerations specified by the user. That is, if a translation command X is issued with a specified speed V_c and acceleration A_c, the DMC will try generate a velocity profile like the one shown in Figure 5.4a, where the area under the curve is the desired translation X. Similarily, if a desired velocity command V_{cm} is issued, the DMC will try to generate a profile similar to the one shown in Figure 5.4b.

In our system the motion of the robot is done by issuing velocity commands. The main advantages of this approach are smooth motions and the fact that motion is regulated according to our best estimate of the state. By having motion based on velocity commands the DMC acts as a slave servo, and the commands are calculated in the controller module using our current estimate of the state given all of our sources of information. On the other hand, motion based on position commands depends on the DMC generating a profile like the one shown in Figure 5.4a, which is executed based exclusively on odometric information

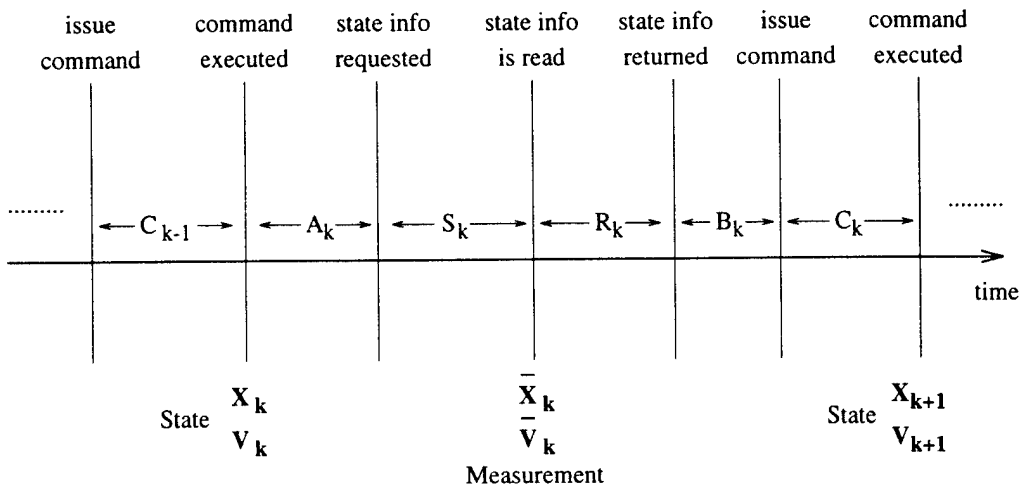


Figure 5.3: Sequence of commands and state information requests.

from the encoders. In this later case we have a single control loop instead of the master-slave configuration of the velocity command approach. The master-slave configuration enables us to compenaste for delays in communication, and to use better estimates of the state than what is possible by exclusively relying on the encoders.

Figure 5.5a shows an ideal velocity profile executed by the robot under different velocity commands with the same specified acceleration A_c. Since the DMC cannot entirely eliminate the robot's dynamics, the real profile looks more like the one shown in Figure 5.5b. For our purposes, however, it is sufficient to assume that the profile approximates the ideal case.

The base of the kinematic model is the following: Assume an initial velocity and position X_o and V_o, and an velocity command V_{cm}. Then, for $0 < t < T$, with $T = (V_{cm} - V_o)/A_c$,

$$X = X_o + V_o t + \frac{A_c t^2}{2}$$

and for the case $t > T$,

$$X = X_o + V_{cm} t - \frac{(V_{cm} - V_o)^2}{2A_c}$$

Referring to Figure 5.3, given position X_k, speed V_k, the prevously (and delayed) velocity command V_{k-1}^{cm}, the next position for $0 < t_k < T$ with $T = (V_{k-1}^{cm} - V_k)/A_c$ is

$$X_{k+1} = X_k + V_k t_k + \frac{A_c t_k^2}{2},$$

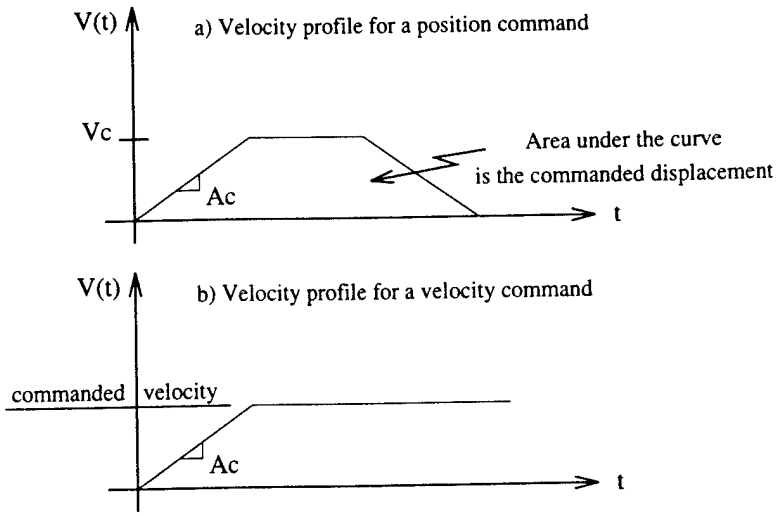


Figure 5.4: Velocity profiles for translation (a) and velocity commands (b).

and for the case $t_k > T$ the next position is

$$X_{k+1} = X_k + V_{k-1}^{cm} t_k - \frac{(V_{k-1}^{cm} - V_k)^2}{2A_c},$$

where $t_k = A_k + S_k + R_k + B_k + C_k$.

Similarily, the measured position $\bar{X}_k$ can be computed. For $0 < \tau_k < T$

$$\bar{X}_k = X_k + V_k \tau_k + \frac{A_c \tau_k^2}{2},$$

and for the case $\tau_k > T$ the measurement is

$$\bar{X}_k = X_k + V_{k-1}^{cm} \tau_k - \frac{(V_{k-1}^{cm} - V_k)^2}{2A_c},$$

where $\tau_k = A_k + S_k$.

Using some intuition about the robot's operation, note that in a continuous operation the velocity commands are not very different from previous ones. That is, the speed increments $V_{k-1}^{cm} - V_k$ are not very large. Thus, by increasing the value of the acceleration A_c, the value of T may be made smaller than $A_k + S_k$ in most cases, and of course smaller than $A_k + S_k + R_k + B_k + C_k$. Under this assumption the kinematic model becomes

$$X_{k+1} = X_k + V_{k-1}^{cm} t_k - \frac{(V_{k-1}^{cm} - V_k)^2}{2A_c},$$

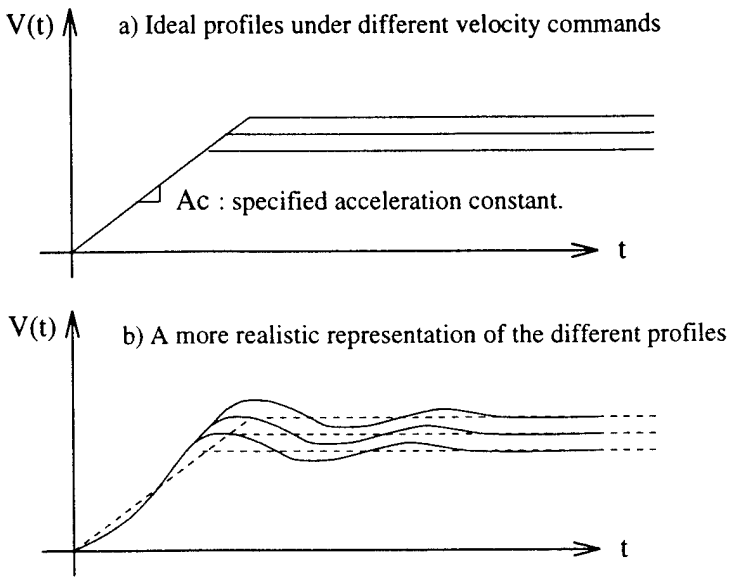


Figure 5.5: Ideal and real velocity profiles under velocity commands.

$$\bar{X}_k = X_k + V_{k-1}^{cm}\tau_k - \frac{(V_{k-1}^{cm} - V_k)^2}{2A_c},$$

where $\tau_k = A_k + S_k$, and $t_k = A_k + S_k + R_k + B_k + C_k$.

Controlling $\bar{X}_k$ is equivalent to controlling X_k. Our goal is to send the measurement $\bar{X}_k$ to zero (sending it to an arbitrary value implies interpreting X_k and $\bar{X}_k$ as the position error and its measurement, so the approach is still general). So the next step is to manipulate the above two equation to obtain a single equation in terms of the measurement $\bar{X}_k$. With some algebra the following equation may be derived:

$$\bar{X}_{k+1} = \bar{X}_k + V_{k-1}^{cm}(t_k - \tau_k) + V_k^{cm}\tau_{k+1} - \frac{(V_k^{cm} - V_{k+1})^2}{2A_c}.$$

This last equation is the *kinematic model of the robot.*

Finding a control law is a compromise between complexity and performance. A simple proportional controller is extremely easy to implement but the closed-loop response will be greatly deteriorated by the delays present in the network. Without any other sort of compensation, the proportional controller will be oscillatory or unstable if the gain is high, and very slow if the gain is low.

With the kinematic model is possible to design a better control law. A preliminary approach, which works well enough in practice, is to assume that all variables change with "smoothness". That is, $\tau_k \approx \tau_{k+1}$, $t_k \approx t_{k+1}$, and the difference between successive command signals

is small compared with A_c. An additional assumption is the fact that usually $(\tau_k/t_k) << 1$. If the conditions of the system are such that the above assumptions constitute reasonable approximations, then it can be shown that a control law of the form

$$V_k^{cm} = -\frac{K}{\hat{t}_k}\bar{X}_k, \quad \forall K > 0,$$

will yield an approximate closed-loop response according to

$$\bar{X}_{k+1} - \bar{X}_k + K\bar{X}_{k-1} = 0,$$

which is a simple second order system, and K may be adjusted to match a particular type of response. In particular, if the response is to be as fast as possible without any overshoot, then $K = 0.25$ will yield a pair of repeated poles at 0.5. A higher value will split the repeated pole into a complex pair (adding oscillations), while a lower will have a slower response. The estimation of $\hat{t}_k$ may be as simple as using the measurement of t_{k-1} as the estimate of the next delay (not recommended), something more elaborate such as the average over all the previuos values of t_k (good approach), or using a weighted average to implement a forgetting factor so the most current measurements have more weight in the estimation (recommended approach).

Of course more sophisticated controllers are possible, but for the sake of brevity which shall omit their description. The advantage of having the kinematic model is that now we are able to design motion control schemes more suitable to our distributed architecture. Our experiments with new controllers will certainly continue.

The Target Error Model

The location of the target is obtained by a vision module that must capture a frame, identify the target, compute distance and angle, and send the information to the motion controller. Since this process is quite involved, the information sent to the controller is already outdated once it is received. The question is how to reconstruct a good measurement based on this.

In Figure 5.6 a simple diagram of the robot and target relative positions is shown. We will assume that all motions consist on pure rotations at this stage, as translation can be regarded as a subsequent problem.

Our objective is to find $\epsilon(t)$, which the angle between the camera line of sight and the target. We assume that the vision module computes a new target position every T seconds on the average, and that the controller cycle is considerably shorter than this. Under this assumption, the controller receives new information as soon as it is ready, and any extra delay is neglible compared with T. This can be expressed as

$$\hat{\epsilon}(t) = \epsilon(t - T).$$

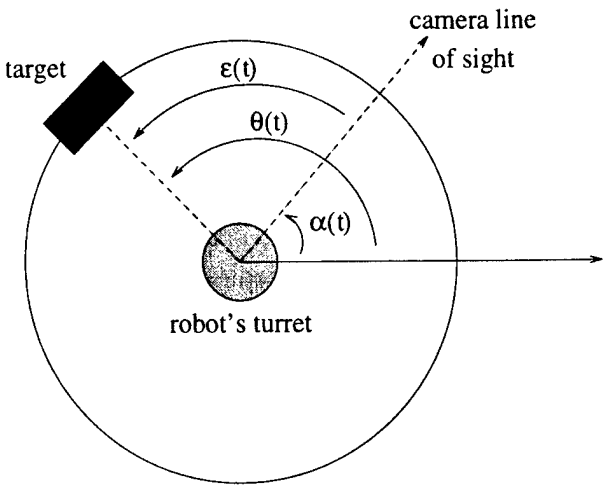


Figure 5.6: Observer and target rotations.

Given that at time $t-T$ the error is changing with speed $\dot{\epsilon}(t-T)$, then the estimate $\bar{\epsilon}(t)$ of the error at time t can be approximated as

$$\bar{\epsilon}(t) \approx \epsilon(t-T) + T \cdot \dot{\epsilon}(T-t), \tag{5.1}$$

$$\approx \hat{\epsilon}(t) + T \cdot \dot{\epsilon}(T-t). \tag{5.2}$$

Then, approximating the derivative of the error with a backward difference we obtain the following:

$$\dot{\epsilon}(t-T) \approx \frac{\epsilon(t-T) - \epsilon(t-2T)}{T}, \tag{5.3}$$

$$\approx \frac{\hat{\epsilon}(t-T) - \hat{\epsilon}(t-2T)}{T}, \tag{5.4}$$

$$\Rightarrow \bar{\epsilon}(t) \approx 2\hat{\epsilon}(t) - \hat{\epsilon}(t-T). \tag{5.5}$$

So the estimate of the target error is based on the current and the previous measurements. In general, the error model may be postulated as

$$\bar{\epsilon}(t) \approx \sum_{i=0}^{m} a_i \hat{\epsilon}(t - iT).$$

The above equation is used when new measurements are obtained, and T must be regarded as the time between successive samples. Although in general T is not constant, we assume that successive intervals are approximately the same (locally contant). The error model was derived for a single instance between samples; since the final equation is independent of T, it can be applied to any successive pair (or set) of measurements.

The time between samples depends on the speed of the vision module. The control loops executes at a faster rate and, in general, data is not ready when it is requested. In this case some prediction about the target error must be done, as the target is moving between samples.

We need to estimate the target angular velocity at the same time we estimate the error. We will assume that this estimate remains valid until the next sample; i.e we are making the assumption that speed is constant between samples. Let δt be the time passed since the last controller cycle, t the current time, ω the angular velocity of the target, and $\alpha(t)$ the position of the turret at time t (which is measurable). From the scheme shown in Figure 5.6,

$$\epsilon(t) = \epsilon(t - \delta t) + \omega \delta t - [\alpha(t) - \alpha(t - \delta t)].$$

Using estimates $\bar{\epsilon}$ and $\bar{\omega}$ in the above equation, we arrive at

$$\bar{\epsilon}(t) = \bar{\epsilon}(t - \delta t) + \bar{\omega} \delta t - [\alpha(t) - \alpha(t - \delta t)], \tag{5.6}$$

which states how the error estimate is corrected every controller cycle if no new information is received from the vision system.

The estimation of $\bar{\omega}$ is done whenever new information is received from the vision module. This is simply

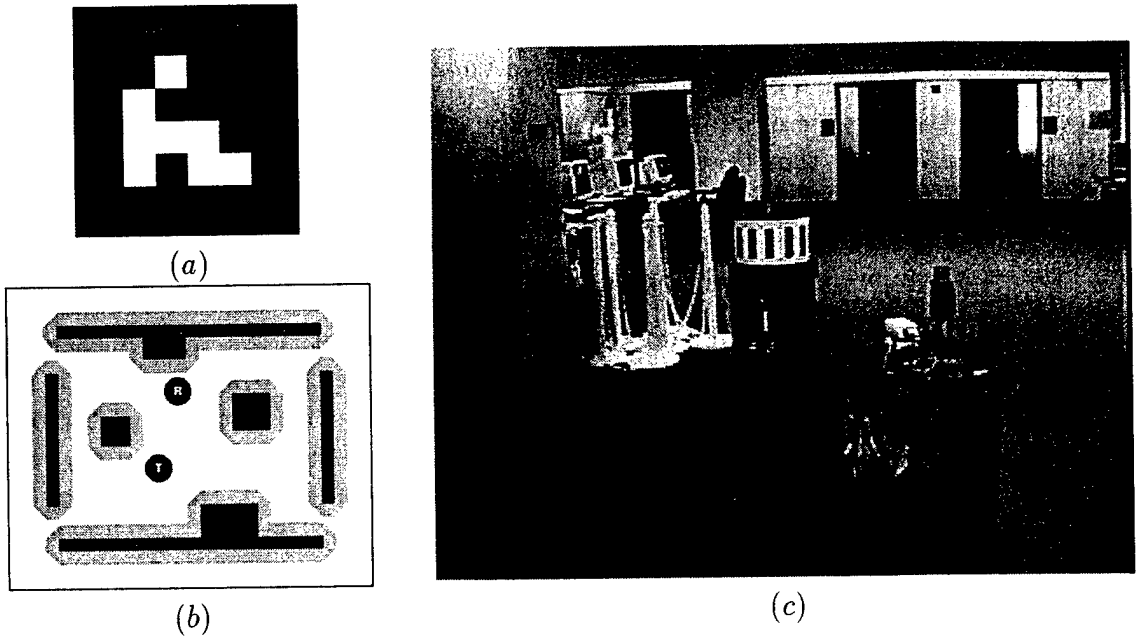
$$\bar{\omega} \approx \frac{\theta(t - T) - \theta(t - 2T)}{T}, \tag{5.7}$$

$$\approx \hat{\epsilon}(t) - \hat{\epsilon}(t - T) + \alpha(t - T) - \alpha(t - 2T). \tag{5.8}$$

The steps to estimate the target angle with respect to the line of sight of the camara are the following:

1. Check if new information is available from the vision module.
2. If new information is available, given the two most recent observations, and the measured time between them, estimate $\bar{\omega}$ and $\bar{\epsilon}$ according to equations (5.5) and (5.8).
3. If no new information is ready, then correct the estimate $\bar{\epsilon}$ according to equation 5.6. This requires a measurement of the change in the turret position during the previous controller execution cycle, the time spent in such cycle, and the most recent estimate of $\bar{\omega}$

The estimate of distance is not critical in our system, so we may in principle use the raw measurements provided by the vision module. However, a minor correction that yields considerable improvement in distance estimation is obtained by subtracting the distance traversed by the robot since the last controller execution cycle to the current distance estimate.



(a) (b) (c)

Figure 5.7: (a) A sample ceiling landmark. (b) The view presented to the user. (c) A view of the IO tracking a second robot.

5.5 Experimental Setup

This section describes our experiments with the autonomous observer prototype. Our experimentation has two goals: first, to validate the utility and robustness of the chosen algorithms for each of the various system components; and second, to demonstrate the feasibility of integrating all of the components into a unified system.

Our experiments took place in our laboratory, a fairly typical office environment. As obstacles we used desks, chairs, and large cardboard boxes. A map, as required by the motion planner, was constructed by approximating the obstacles by polygons; this map is shown in Figure 5.7.a.

The observer itself is a NOMAD-200 robot with an on-board Pentium-based computer. It is equipped with an upward-pointing camera for landmark detection and a forward-pointing camera for target tracking. Both cameras are mounted rigidly to the robot's turret, which can rotate independently of its drive wheels. This allows the turret (along with the tracking camera) to rotate based on the motions of the target without affecting the motion of the robot.

As a target, we used a second NOMAD-200 equipped with a special "hat" required by our

simplified tracking algorithm, as described above. The target was moved under joystick control by a human operator. Figure 5.7.c shows a view of both the target and the observer.

As stated above, the purpose of our experiments was to verify each of the components in the system, as well as to show that they could be successfully integrated. We have been able to quantify the performance of the individual of components. Furthermore, the system can be successfully integrated: the observer was consistently able to visually track the target and keep it in view in the presence of obstacles.

The most important conclusion of our experimentation is that, even though each of the components which makes up the system may periodically fail, the overall system is robust in the face of those failures. For example, the target tracker or landmark detector may periodically give an inaccurate result, but these errors are tolerable since they are random variables with zero mean and a relatively small variance. In addition, the sampling rate is high relative to the reaction time of the control system, so sporadic sensing failures do not significantly alter transient responses of the system.

5.6 Discussion

In this chapter we have described the high-level design and implementation of an autonomous observer. We have also described our initial prototype as well as early experiments with the system as a whole. Up to this point, our time has been spent designing an integrated system, not in refining and optimazing the individual components. Our continuing experimental work focuses on two types of extensions: First, those which increase the generality and robustness of current components; and second, those which add new functionality to the concept of the overall autonomous observer system.

In terms of improving upon the current components, we have the following goals. First, we are working on ways to remove the discretization in our planning implementation by developing new visibility algorithms more directly related to the view planning problem. Second, we are working on ways to allow the planner to look farther ahead in time when it considers possible target moves. Third, we are working to implement a more general pattern-tracking mechanism which removes the need for a special visual cue on the target.

In terms of adding functionality to the system, we are working on several extensions. First, we plan to add the automatic 3D model construction component, a subsystem that has already been tested individually. The model would then be used to reconstruct a view for the user. Second, we plan to extend the view planner to deal with multiple observers which can cooperate to track a single target. Finally, we will also consider the case of multiple targets and the problem of dynamically assigning observers to these targets.

Appendix A

Landmark-Based Navigation

A.1 Introduction

The problem of reliable navigation is among the most pervasive in all of mobile robotics. For a robot to be truly mobile, it must be able to repeatably move from point to point while keeping track of its current location with respect to its environment and robustly recognizing when it achieves its goals. Although this problem has received considerable attention, many existing systems often lack flexibility, reliability, or both. Computational cost is also an issue. The main problem is that of dealing with uncertainty, which refers to the statistical distribution of errors in both control and sensing.

We have chosen to use simple, artificial landmarks as a means of limiting the control and sensing uncertainty in the robot's environment. We claim that such landmarks require only a small amount of environmental engineering, and that they make it possible to build a very flexible and robust navigation system. The notion of a landmark is not new and its role in robot navigation has been previously discussed in several papers, including [34, 43, 51, 69]. Here we use this notion in a more formal and systematic way.

Section 2 defines a navigation problem which embeds enough assumptions to make motion planning polynomial. Section 3 describes how the required assumptions can be enforced in the real world using simple, artificial landmarks. Section 4 reports on experiments performed to validate our algorithms and to verify that the engineering costs were reasonable.

A.2 The Navigation Problem

In this section we describe a motion-planning problem which, using the assumptions enforced by landmarks, admits a polynomial-time solution. We then discuss ways in which a planner can be used as part of a landmark-based navigation system.

Problem statement: We represent the mobile robot as a point moving in a plane among obstacles which are represented as generalized polygons (regions bounded by straight edges and circular arcs). Conveniently, this corresponds exactly to the case of an omnidirectional circular robot moving among generalized polygonal obstacles if we shrink the robot to its centerpoint and grow all of the obstacles by its radius.

We let $\mathcal{W}$ denote the free space in which the robot can move, called the *workspace*. The navigation problem is to find and execute a motion that causes the robot to move from some initial region (a subset of $\mathcal{W}$) to some goal region (another subset of $\mathcal{W}$).

Although we assume (for the purposes of planning only) that the geometry of $\mathcal{W}$ is perfectly known, we allow for some uncertainty in both robot control and sensing. We model control uncertainty by assuming that control error is bounded by some angle $\theta < \pi/2$. This means that when the robot is commanded to move in a direction d, it will actually move in some unknown direction d', but d' will differ from d by less than θ. We model sensing uncertainty by defining a region $\mathcal{U}$ such that, if the robot's actual position is q, then its sensed position lies within $\mathcal{U}(q)$. For example, $\mathcal{U}(q)$ may be a disk centered around the point q.

So far, we have not made any assumptions that let us bound sensing and control uncertainty. Without such assumptions, problems very similar to this one have been proven hard in [12, 57].

The role of landmarks: To make the problem computationally tractable, we will introduce the notion of *landmark regions* (see [50]). Each such region is denoted by L_i and is a subset of $\mathcal{W}$. We define landmark regions such that sensing and control uncertainty are bounded within them as follows:

1. A robot entering landmark region L_i will reliably sense that it is within L_i, although it will not know precisely where it is within L_i.
2. Once in L_i, a robot can reliably navigate into a predefined subset of L_i, called the landmark region's *kernel*. The robot will not know exactly where within the kernel it is located.

Intuitively, a landmark region is some subset of the workspace from which a robot can sense some distinctive feature (a landmark). It can then use this sensing to localize itself within some smaller region (the kernel). Outside of landmark regions we assume nothing about the sensing uncertainty; in fact, we do not even assume that useful sensing exists. Note that if this is not the case, then sensing outside of landmark regions could be used to reduce the control error θ.

One important property of landmark regions as defined here is that *they do not require perfect control or sensing anywhere in the workspace.* They do, however, define a set of states which the robot must be able to reliably recognize. Previous work [14, 21, 24, 22] yields substantial evidence that planning is exponential when state identification is not straightforward (e.g., when it relies on sensing history). Landmark regions and their kernels provide a predefined set of pertinent states which are directly identifiable by a robot's sensors.

Work described in [50] shows that, using only the assumption of landmark regions as defined above, it is possible to build a motion planner that is sound and complete yet runs in polynomial time. Using such a planner in an on-line fashion makes it possible to plan motions from a given initial state to a desired goal state which are guaranteed to be reliably executed.

Validating landmark placement: Two problems arise when placing landmarks into a particular workspace. First, one must determine how many landmarks are needed. Second, one must determine where these landmarks should be placed such that the robot will be able to perform any foreseeable task.

A planner such as the one described in [50] may be used to evaluate a given placement of landmarks. While planning paths between landmark regions, the planner makes sure that each landmark region is reachable from any other by some set of motions which are guaranteed to be reliable. Failures to find plans in some areas of the workspace suggest that new landmarks must be created in those areas.

In some cases it may be preferable to use only a weak motion planner (or none at all) during execution. Such a planner is not guaranteed to produce totally reliable plans, but these plans are ofter faster to execute. Even without reliable plans, navigation can be made reliable if enough landmarks are used and they are placed appropriately. Here an evaluation tool is extremely useful since it allows the verification of a landmark arrangement without requiring trial-and-error experimentation.

A.3 Landmarks

In the formulation of the navigation problem we make assumptions that bound the sensing and control error within certain parts of the workspace. In this section, we describe how we use landmarks to engineer the environment to enforce these assumptions. These landmarks are the "distinctive features" that induce landmark regions where the robot can use sensory data to localize itself with bounded error.

In our work we have chosen to place artificial landmarks into the environment, rather than relying on natural landmarks such as walls or corners. The main motivations for this are speed and reliability: we can specifically design artificial landmarks that can be quickly and reliably sensed and which are also "low-cost" in that they do not require much engineering of the environment. Without placing our own landmarks into the environment, we would be faced with the much more difficult task of detecting whatever assortment of features happened to be present already, and there would be no guarantee that these landmarks would be sufficient to complete any particular task.

In general, a landmark must be designed so that a sensing system can achieve three key functions:

1. *Detection* The robot must be able to quickly determine whether or not there is any landmark in its field-of-view.

2. *Localization* The robot must be able to accurately determine its position in the workspace relative to the landmark.

3. *Recognition* The robot must be able to differentiate between the different landmarks in the workspace. Once a landmark is identified, the robot can localize itself relative to the workspace if the landmark's position is known.

It is important to note that there are several different scenarios in which landmarks are useful. Consider, for example, an office delivery task: here, landmarks would be useful not only for navigation but also to mark objects to be delivered. As another example, consider the task of automatic map generation: known landmarks could define fixed reference points during the map-building process. Upon detecting a previously-unknown landmark, the robot could use those reference points to localize the new landmark with a high degree of accuracy.

We have experimented with two types of landmarks. The first is mounted on the ceiling and detected with an upward-looking camera. The second is detected with a forward-looking pair of cameras and can be placed on walls or any other "eye-level" surfaces. For each case we describe the methods used to achieve the three basic functions outlined above.

Figure A.1: A sample ceiling landmark

A.3.1 Ceiling Landmarks

Figure A.1 shows a sample of one of our ceiling-based landmarks. The landmark pattern consists of a black square with a 4×4 pattern of smaller squares inside of it. The orientation of the landmark is done by detecting the bounding box of the pattern. The slopes of the edges determine the orientation of the landmark, while their intersections locate the corners. Once the landmark is localized, the positions of the inner squares are computed and their intensities are read. These intensities are grouped into "black" and "white" subgroups to determine the 16 binary values they represent. Four of these values are used to disambiguate the landmark's orientation, and the remaining 12 encode the landmark's unique ID (giving $2^{12} = 4096$ possible distinct landmarks). The landmarks are placed at well-known positions in the workspace.

Detection: While moving, the robot continually captures images with its upward-pointing camera. Each image is grayscale with a resolution of 256×243 pixels. As a first step, the image is binarized using a global threshold which is determined based on a histogram of the intensity levels in the image. The detection algorithm then identifies edge pixels in the image using a variant of the algorithm presented in [15], and extracts edge chains that are consistent with the boundary of a square. When such a chain is found, the corners are detected and lines are fitted to the pixels which make up each edge. Slope information is available after the line-fit step, and the orientation of the landmark can be then computed.

Localization: A landmark's position relative to the robot has three parameters: x, y, and θ. In our case, x and y are computed by finding the center of the landmark in the image and then mapping those values into the workspace coordinate system using the parameters of the robot's camera. The value of θ is computed by using the slopes extracted in the detection phase.

Recognition: Given the location and orientation of the landmark in the image, we calculate the expected positions of its 12 identifying tiles. The intensities are read from the image using bilinear interpolation. Accuracy may be increased by sampling several pixels from each tile. An error-correcting encoding could also be used, but this has not proven necessary in our experiments.

Analysis: We designed the landmarks so that the detection algorithm could be as simple as possible. Because it is so simple, it is also fast; the entire algorithm runs in less than 15 milliseconds on a desktop workstation. The time required to detect a landmark is important because it affects the speed at which the robot can move. If the robot moves too far during the time between the acquisition of subsequent frames, there will be a section of ceiling which is not captured in either frame. Landmarks in this area will not be detected, causing navigation to fail.

Each landmark induces a circular region C in the workspace from which the pattern is guaranteed to be visible. However, since sensing is not instantaneous, the robot may pass through C on a path nearly tangent to its boundary and miss the landmark pattern because it is busy processing a previous image. This leads us to define a landmark region L concentric to C and ϵ smaller in radius. The value ϵ is derived from the velocity of the robot and the time required to process an image. In this way we guarantee that if the robot travels along a path tangent to L it will always acquire an image while within C.

We have performed many experiments with these landmarks under normal office conditions. Our landmarks had a size of 12×12 inches and were placed 9-11 feet above the camera. Detection and identification have proven to be totally reliable provided that landmarks are placed appropriately to insure that lighting conditions are relatively consistent when the robot is viewing the various landmarks.

The algorithm is very accurate: the translational error has zero mean and a standard deviation of 0.75 inches, while the rotational error has also zero mean and has a standard deviation of 0.5 degrees. These results are limited by the image resolution and the time spent on image analysis.

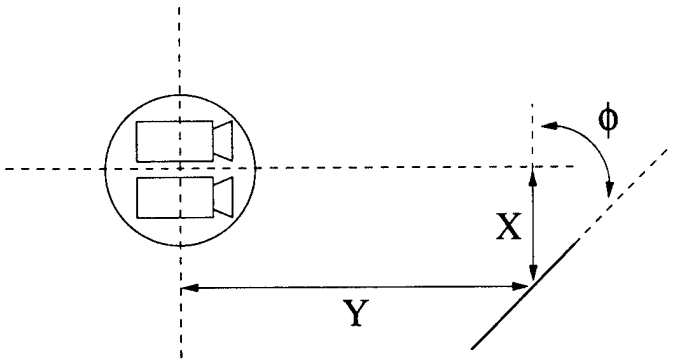


Figure A.2: The parameters x, y, and ϕ which describe the landmark's position relative to the robot

A.3.2 Wall Landmarks

Physically, the landmarks we use on walls are nearly identical to those we use on the ceiling. This time, however, it uses a stereo pair of forward-looking cameras.

To recognize wall landmarks we use an algorithm based on moment invariants. In [33], Hu proposed the use of geometric moments of binary patterns to extract a set of coefficients which could be used to recognize those patterns. Reiss showed in [65] that these descriptors are invariant under general affine transformations. This invariance is important in our application, since the robot-mounted cameras may be at any distance from the plane containing the landmark, and at any angle relative to that plane.

Detection: Our algorithm first detects the edges in each image using a common operator such as those described in [56]. It then extracts chains of edge pixels. If there is a landmark pattern in the image, its boundary will be found in this manner. It then computes estimates of various moments from each set of boundary pixels, as described in [18]. Comparing these estimates with the expected values determines whether or not a landmark has been detected.

Localization: In this case there are also three parameters that describe the landmark's location relative to the robot: x, y, and ϕ (see Figure A.2). We first locate the center-of-mass of the landmark in each camera image and compute x and y using stereo correspondence. Next, we compute ϕ by analyzing the moments to reconstruct the affine transformation between the image plane and the landmark plane.

Recognition: As in the ceiling-based case, given the results of localization we are able to predict the locations in the image of the landmark's 12 tiles, and directly sample their values.

Analysis: The speed of this algorithm compares well with the algorithm used to detect ceiling landmarks. We have conducted many experiments to determine the algorithm's accuracy in terms of robot localization. The distance from the robot to the landmark is recovered with an error of less than 10%. The orientation of the landmark plane relative to the image plane is measured with an error of less than 15 degrees.

One benefit of wall-based landmarks is the ability to sense them from a relatively large area. Using our current hardware, we can recognize landmark patterns at distances of up to 20 feet. The accurate measurement of the landmark's orientation, however, is possible only at distances up to 14 feet.

A.3.3 Comparison of Landmarks

We have considered various types of landmarks, but have found the two presented above to be the most useful. In this section, we discuss the general tradeoffs between ceiling- and wall-mounted landmarks.

We have found that ceiling-based landmarks are especially useful in navigation tasks. The placement of ceiling landmarks and the position of the robot have the same degrees of freedom; this means that the image of a landmark are simply and directly related to the position of the robot. Although the area in which a landmark is visible might be small, its shape is well-understood and sensing error does not vary appreciably within that region. The localization uncertainty is small and easily characterizable.

One difficulty with ceiling-based landmarks lies in finding appropriate placements for them. This is especially true outdoors, where ceilings are uncommon. In addition, our algorithms are sensitive to drastic brightness differences within images, so we cannot place landmarks close to the overhead flourescent lights in our lab. More sophisticated algorithms would solve this problem, but have proven to be intolerably slow on our robots. Also, placing landmarks over uneven floors results in significantly skewed images and, therefore, bad localization results.

Wall-based landmarks, however, have a complementary set of advantages and disadvantages. First of all, they are more general than ceiling-based landmarks in that they can be used to mark arbitrary objects in the robot's environment. One example is a delivery task; deliverable objects could each be labelled with a unique landmark. They are also generally

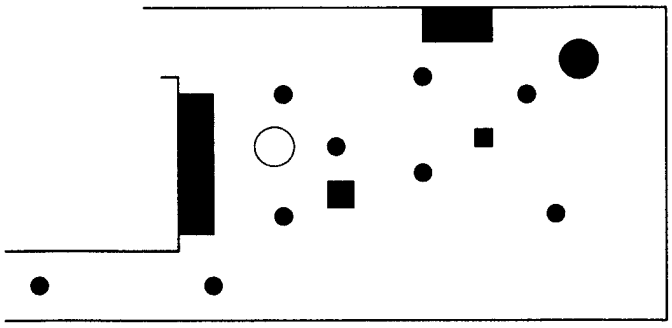


Figure A.3: The workspace for experiments with ceiling landmarks

visible from a relatively large area, which in many cases could mean that fewer landmarks are required.

One problem with wall-based landmarks is that, because the algorithms used to detect them can make fewer assumptions, they also tend to be less accurate. The impact of this can be reduced, though, if the robot is allowed to track the landmark over time, keeping the pattern in sight and using it as a "beacon" to guide it to its destination. Another potential problem is that other objects in the workspace—such as chairs, desks, and moving people—tend to occasionally occlude wall-based landmarks. This makes it difficult to reliably describe the area from which such a landmark should be expected to be visible. In the case of ceiling landmarks, such unforeseen obstructions are far less likely.

A.4 Experimental Results

All of our experiments were performed with a NOMAD-200 mobile robot from Nomadic Technologies, Inc. It consists of a nonholonomic, zero-turning-radius base which supports an independently-rotating turret. On-board computation is provided by an Pentium-based computer running at 90 MHz, which was used to run all of the image-processing and navigation software.

Experiments with ceiling landmarks: In this first set of experiments we combined the planner presented in [50] with the ceiling landmarks described above. The landmarks induce landmark regions L_i which—along with descriptions of the workspace $\mathcal{W}$, initial region $\mathcal{I}$, and goal region $\mathcal{G}$—are the inputs to the planner. Each landmark induced a landmark region with a diameter of 16 inches.

The experiments were performed in our laboratory space, which is an office-like environment

with rooms, hallways, and a typical selection of furniture. Figure A.3 depicts a subset of that space consisting of a large room and a narrow hallway. Obstacles (walls, tables, chairs, a trash can, and an idle robot) are shown in black. Landmark patterns are depicted in gray. To provide a sense of scale, the robot is shown in white at the same scale; it has a diameter of 24 inches.

Experimentation showed that our complete navigation system is extremely reliable. The system is consistently able to navigate from landmark region to landmark region for the entire battery life of the robot (approximately one hour). The system has also been used with similar success in a second office environment with different carpeting and lighting.

Experiments with wall landmarks: This set of experiments took place in a 3000-square-foot warehouse. Mobile partitions were used to create an office-like layout with fourteen rooms and several hallways. In contrast to the experiments described above, here we did not use landmarks as a basis for planning and navigation but for two entirely different tasks.

In the first case, the robot's task was to explore the environment and look for landmarks. Landmarks it found were added to an internal geometric map of the environment. During this time, the robot planned its exploration by simply searching a graph which represented a topological map of the workspace.

Once a map had been generated, the second task was to find and deliver[1] objects. Each object was assumed to have been marked with a unique landmark. When provided with a target object and a goal position, the robot used its previously-built map to move close to the target object. Once it was in range, it used the object's landmark as a beacon and approached the object accurately. It then moved to the goal landmark in a similar fashion.

Once again, these experiments validated the robustness of our approach. The robot consistently mapped the landmarks with enough accuracy to then carry out the delivery task successfully.

[1]Our robot has no way to pick up objects, so delivery consisted of moving near an object, identifying it, and then moving to the goal location.

Appendix B

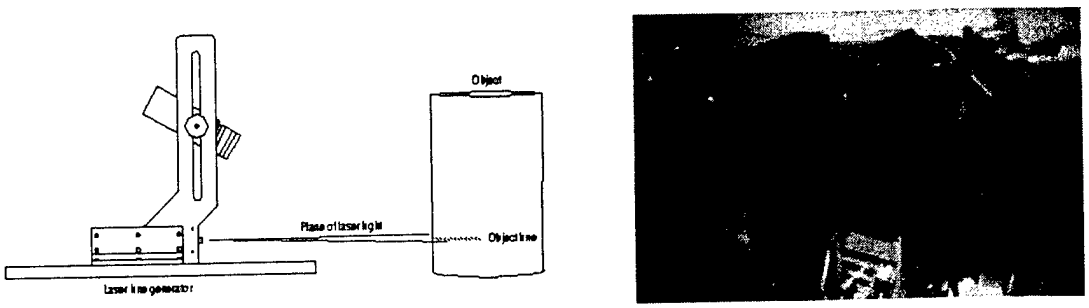
Range Image Acquisition

B.1 Laser Range-Finder Model

The mobile platform we used for our experiments in model construction is a Nomad 200 robot from Nomadic Technologies. The robot is equipped with a laser range-finder sensor, which consists in a solid state laser, a CCD camera, and a signal processing card named the Sensus 500 board. The range-finder operates under the principle of triangulation, the laser projecting a plane of light perpendicular to the CCD retina. An schematic view of the sensor is shown in Figure B.1a, and a photograph of the actual configuration on top the robot is shown in Figure B.1b.

The CCD camera is equipped with a red filter, which ideally blocks all light frequencies except the laser fundamental frequency. The Sensus 500 board detects the pixel of highest intensity values in each row in the image. One row is called a *scanline*, and the collection of all the intensity peaks for each scanline is called a *scan profile*. In order to make sure that the intensity peak for one scanline is not a spurious event, the peak is counted only if its intensity is above a certain threshold and if the width of the peak is below a tolerance. This processing is done by hardware on the Sensus 500 card.

In the end, the data coming out of the card consists of an array of 486 elements (the number of rows in the CCD), containing the column numbers of those pixels with the highest intensity value for each row, or a value of -1 in case no peak could be confidently detected. This array, which we call $\mathcal{A}$, defines a scan profile. A collection $(\mathcal{A}_k)_{k=1}^{n}$ defines a *sweep*, which is the result of rotating the turret supporting the range finder. The calibration and modeling problem consists in translating the integer values in $\mathcal{A}$ into actual physical distances.



a *b*

Figure B.1: (*a*) A schematic view of the range-finder sensor. (*b*) A picture of the actual sensor oriented horizontally so as to scan vertically (the middle camera is for color acquisition).

B.1.1 Projection Model

Figure B.2 shows the laser-camera configuration from a viewpoint parallel to the laser plane of light. The laser is projected from left to right, and we define this direction the direction of increasing x. We define the direction into the picture as the direction of increasing y. The CCD camera is placed at an orientation perpendicular to the plane of light, with its top in the direction of y. The camera's reference plane is oriented at an angle α with respect to x, and its line of sight intersects at a distance h from the point A. This intersection will be the origin of the axis x. The origin of the axis y will be the center of the plane of light.

A point (x, y) is transformed into reference C (a frame parallel to the camera reference) according to

$$x_c = x \cos \alpha, \tag{B.1}$$

$$y_c = y. \tag{B.2}$$

The frame C is at a distance $z = (x + h) \sin \alpha$ from the camera reference F. Hence, a point (x, y) is transformed to the camera reference as

$$x_f = f \frac{\cos \alpha}{\sin \alpha} \frac{x}{x + h}, \tag{B.3}$$

$$y_f = f \frac{1}{\sin \alpha} \frac{y}{x + h}, \tag{B.4}$$

where f is the focal radius of the camera. Finally, the frame F is transformed to pixel values according to

$$x_p = k_x x_f + c_x, \tag{B.5}$$

$$y_p = k_y y_f + c_y, \tag{B.6}$$

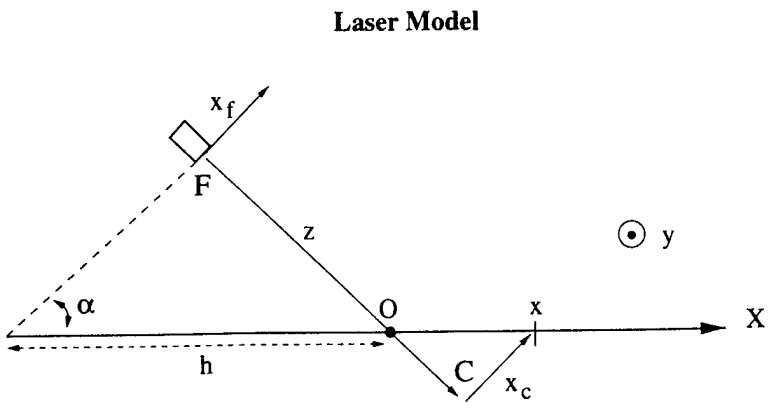


Figure B.2: Laser-camera configuration of the range-finder. F is the camera reference, and C is a local reference parallel to F. The origin O is set at the intersection of the camera line-of-sight with the laser plane.

where k_x, k_y, c_x, c_y are some constants. The lumped model will be

$$x_p = \frac{x}{hd + dx} + c_x, \text{ with } d = \frac{1}{k_x f}\frac{\sin\alpha}{\cos\alpha} \tag{B.7}$$

$$y_p = \frac{y}{hb + bx} + c_y, \text{ with } b = \frac{1}{k_y f}\sin\alpha \tag{B.8}$$

which expresses the pixel values in terms of the physical coordinates (x, y). The converse operation is given by

$$x = \frac{dh(x_p - c_x)}{1 - d(x_p - c_x)}, \tag{B.9}$$

$$y = \frac{bh(y_p - c_y)}{1 - d(x_p - c_x)}. \tag{B.10}$$

B.1.2 Noise

It is interesting to note that the main uncertainty present in the sensing process is during the detection of the pixel with highest intensity value for each row in the image. That is, for row y_p the sensor returns a peak position at $\hat{x}_p(y_p) = \text{round}[x_p(y_p) + \omega]$, where $x_p(y_p)$ is the real column location of the peak at row y_p, and ω a random variable of mean zero and variance σ^2 (here is assumed that the noise is not dependent in the row number). Hence, a scan profile is a sequence of pixel points $(\hat{x}_p^i(y_p^i), y_p^i)_{i=0}^m$ which is uniformly increasing in y_p (unit increments), with random (noisy) column quantities $\hat{x}_p^i$. When the pixel information

is converted into physical distances by equations (B.9) and (B.10), then both x and y coordinates are corrupted by noise, since both depend on x_p. Furthermore, none of the physical coordinates are necessarily monotonically increasing.

However, if we define the coordinates ν and ψ as

$$\nu = \frac{x}{x+h}, \tag{B.11}$$

$$\psi = \frac{y}{x+h}, \tag{B.12}$$

then equations (B.7) and (B.8) imply that

$$\nu = d(x_p - c_x), \tag{B.13}$$

$$\psi = b(y_p - c_y). \tag{B.14}$$

Thus, given a sequence of pixel points $(\hat{x}_p^i(y_p^i), y_p^i)_{i=0}^m$ which is uniformly increasing in y_p, then $(\hat{\nu}_i, \psi_i)_{i=0}^m$ will be uniformly increasing in ψ. Furthemore, if $\hat{x}_p$ is the noisy component of a pixel pair, then $\hat{\nu}$ will be the only noisy component of the pair $(\hat{\nu}, \psi)$.

Coordinate ν can be roughly interpeted as a re-scaling of the physical quantity x. Coordinate ψ, on the other hand, can be loosely related to tangent of the angular element of a polar coordinate pair.

B.1.3 Calibration

The model described in the previous subsection requires identification of five parameters. Of these, one can be indirectly obtained accurately, while the others must be estimated using a calibration procedure.

First, let us consider h. Although it is difficult to establish the intersection between the camera line of sight and the laser plane of light, measuring the angle α and the distance $h\cos\alpha$ is relatively easy. Given the dimensions of the system, the errors carried from not establishing the camera center precisely are not significant. With a minimum of care, it is possible to obtain an estimate of h of reasonable confidence.

The parameters c_x, c_y, b, d cannot be measured directly. Some preliminary experiment is required. Our experiment consisted in orienting the range-finder vertically, such that the plane of light is parallel to an horizontal optic table (see Figure B.3a). On the optic table, the sharp corner of a box was placed at regular intervals, spanning the table at evenly spaced grid locations. The range-finder was oriented so its x and y axes (from Figure B.2) were aligned with the grid in the optic table. At each grid position, a scan profile was acquired

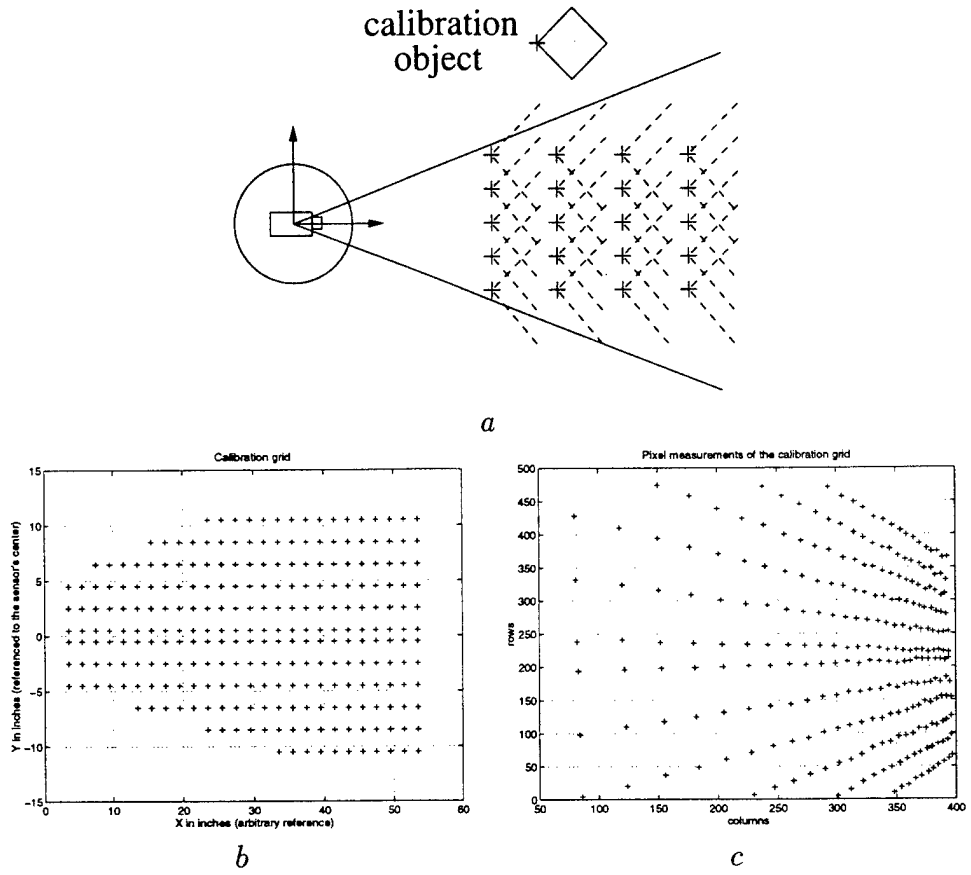


Figure B.3: (a) Calibration table, samples are taken at uniform intervals. (c) Calibration grid. (c) Pixel values of the calibration samples.

and the pixel coordinates corresponding to the box's corner were identified. At the end of the acquistion process, we ended up with an array of points evenly spaced in the physical coordinates (x, y) and a collection of pixel values associated to them. The grid (with arbitrary origin), and its pixel equivalent, is shown in Figure B.3b and B.3c.

The projective model tells us that physical locations along constant values of x are transformed onto a lines of constant values of x_p. However, lines of constant y are not transformed to lines of constant of y_p. From Figure B.3b we can see graphically how the warping takes place. It is apparent that lines of constant y transform to radial rays projecting into some sort of center.

Some further manipulations of the model equations can describe analytically how a uniform

grid is warped. From equations (B.7) and (B.8):

$$y_p = \frac{h}{x+h}\frac{y}{b} + c_y, \tag{B.15}$$

$$\frac{h}{x+h} = 1 - d(x_p - c_x), \tag{B.16}$$

which imply that

$$y_p - c_y = \frac{yd}{hb}\left[\frac{1}{d} + c_x\right] - \frac{yd}{hb}x_p. \tag{B.17}$$

This last equation tells us that lines of constant y are transformed into lines of slope $-(yd)/(hb)$ in the pixel coordinates. Furthermore, equations (B.15) and (B.16) tell us that if $x \to \infty$, then $x_p \to 1/d + c_x$ and $y_p \to c_y$.

So the first step in our calibration procedure consists in estimating the line parameters for each ray shown in Figure B.3b, which can be done with a standard least-squares procedure. Given r number of rows in the grid, we will end up with $(m_1, \ldots, m_r)$ and $(c_1, \ldots, c_r)$ as the line parameters of the r rays.

We do not know the values of y in the grid, but their spacing. Given two different horizontal grid locations y_j, y_i, with known spacing $\delta_{ij} = y_j - y_i$, we can combine the equations of the two corresponding rays. This results in

$$y_p = \frac{\delta_{ij}d}{hb}\left[\frac{1}{d} + c_x\right] - \frac{\delta_{ij}d}{hb}x_p, \tag{B.18}$$

$$y_p = (c_j - c_i) + (m_j - m_i)x_p, \tag{B.19}$$

which imply that $(m_j - m_i) = \beta\delta_{ij}$, with $\beta = d/(bh)$. Given that we have r rows, we can combine $r(r-1)/2$ distinct pairs of rays. Let $\Delta = [\delta_{ij}]$, $\forall i, j$ such that $i = 1, \ldots, r-1$, $j = i, \ldots, r$; similarily define $\delta_m = [m_j - m_i]$. Then, the least square estimate of β is given by $\hat{\beta} = (\Delta'\Delta)^{-1}\Delta'\delta_m$. We thus obtain the estimation of $d/(bh)$.

We can also estimate the intersection coordinates of all the rays. If two distinct rays intersect at (x_c, y_c), equations

$$y_c = m_j x_p + c_j, \tag{B.20}$$

$$y_c = m_i x_p + c_i, \tag{B.21}$$

imply that

$$(c_i - c_j) = (m_j - m_i)x_c, \tag{B.22}$$

$$(m_j c_i - m_i c_j) = (m_j - m_i)y_c. \tag{B.23}$$

Again, using all possible distinct pairs of equations we may obtain up to $r(r-1)/2$ conditions to estimate x_c and y_c using least squares. Once the intersection center is estimated, $c_x+1/d \approx \hat{x}_c$ and $c_y \approx \hat{y}_c$.

Given estimates for c_y, $c_x + 1/d$, and $d/(bh)$, we can find the individual estimates if we are able to find a fourth independent estimation. This can be done from equation (B.15):

$$y_p(x,y) = \frac{1}{b}\frac{y}{x+h} + c_y. \tag{B.24}$$

For constant values of $x = x_k$ (one column in the grid), and distinct values of y, we obtain

$$\begin{array}{rrcl} & y_p(x_k,y_j) & = & \frac{1}{b}\frac{y_j}{x_k+h} + c_y, \\ & y_p(x_k,y_i) & = & \frac{1}{b}\frac{y_i}{x_k+h} + c_y, \\ \rightarrow & y_p(x_k,y_j) - y_p(x_k,y_i) & = & \frac{1}{b}\frac{y_j-y_i}{x_k+h}, \\ & \delta_{yp}(x_k;j,i) & = & \frac{1}{b}\frac{\delta_y(j,i)}{x_k+h}. \end{array} \tag{B.25}$$

Now, taking different values of x (different columns in the grid), and combining pairs of equations:

$$b\delta_x(k,l) = \frac{\delta_y(j,i)}{\delta_{yp}(x_k;j,i)} - \frac{\delta_y(j,i)}{\delta_{yp}(x_l;j,i)}. \tag{B.26}$$

This last equation takes the difference at two distinct x locations in the grid of the ratio between the spacing in y and the spacing in pixel rows ($\delta_y(j,i)/\delta_{yp}(x_k;j,i)$), and relates it with the spacing in x. Given all the possible combination between differences in y with differences in x, we can get a number of conditions roughly equal to $1/4$ of the square of the total number of points in the grid. We use these conditions to estimate b using least squares.

Due to noise in the measurements, it is convenient to try an approach similar to the one used to estimate $d/(bh)$ before using equation (B.26). In order to estimate $d/(bh)$, we first fitted line rays to each group of pixels originated from the same row in the grid; with the line parameters of these rays we computed our estimation. Similarily, in order to compute the estimate $\hat{b}$ using equation (B.26), we may first fit a line to each group of pixels coming from the same column in the grid. These lines should be vertical lines, given our proposed experimental setup. Afterwards, we find the intersections of these lines with the rays previously computed, and the intersection points should be the "pixel" values used to compute the differences in equation (B.26). Figure B.4 shows an example of the resultant line-fits.

The calibration procedure is summarized as follows:

1. Mount a setup such as the one shown in Figure B.3. Measure the distance $h\cos\alpha$ and the angle α in order to indirectly measure h (see Figure B.2). Define a uniform grid and take measurements with the range-finder at each grid location.

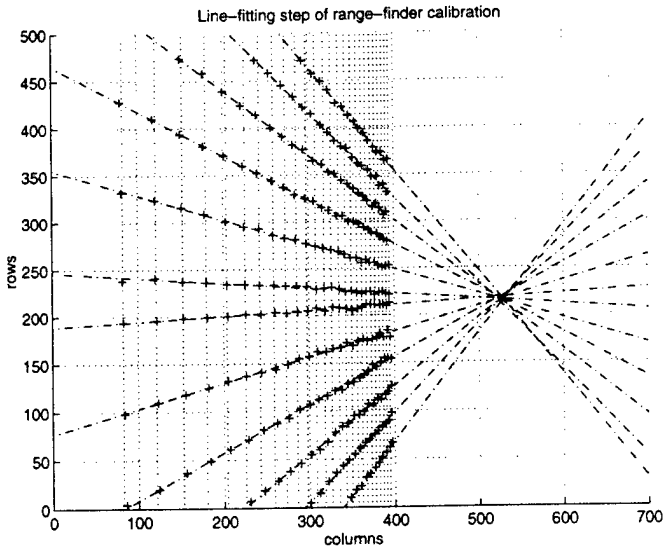


Figure B.4: Line-fitting step of the calibration procedure. Diagonal rays intersect at point $(c_x + 1/d, c_y)$, the pixel equivalent of objects located an infinite distance ahead of the sensor.

2. Group the pixels according to the rows in the grid they originated from. Fit lines rays to each of the groups. These lines should appear as radial rays projecting into a center at the right of the image (see Figure B.4).

3. Compute the estimate of $d/(bh)$ by least squares, using the conditions generated by equation (B.19). Similarily, compute estimates of (x_c, y_c) by least squares using equations (B.22) and (B.23); then $c_x + 1/d \approx \hat{x}_c$ and $c_y \approx \hat{y}_c$.

4. Group the pixels according to the columns in the grid they originated from. Fit lines to each of the groups. These lines are practically vertical (see Figure B.4).

5. Compute the estimate of b by least squares, using the conditions generated by equation (B.26). Use the intersections of the lines computed in the previous step with the rays computed in the second as the pixel values used to calculate the differences required by the equation.

6. Given the estimate of b, obtain the approximations to d, c_x, and c_y. The model is now complete.

Note that the laser is calibrated according to the origin defined in the previous subsection (see Figure B.2). Values of y are centered with the sensor, but the origin of x is not necessarily the sensor. Its is very easy, however, to shift the origin of x by adding an offset. If d_o is the

distance to the sensor for an arbitrary grid point, and x_o the actual measurement produced by the calibrated model, then the offset $(d_m - x_o)$ will place the origin at the sensor.

B.2 Fast Piecewise Linear Approximation of Scan Profiles

As explained in Chapter 2.2.1, the input to the second and final stage of the profile segmentation process consists of data points grouped into clusters (see Figure 2.3). Each cluster describes a continuous portions of the curve $r(\theta)$, which if taken as a whole is not necessarily continuous function (see Figure 2.4). We now want to represent a cluster of points $\{\rho_i, z_i\}$ as succession of segments, which is our choice of geometric primitives.

Fitting a sequence of segments to a data set can be seen as a problem of piecewise linear approximation. Algorithms to solve this problem are well studied, and given an appropriate domain parition they are relatively straightforward to implement. The problem arises when we are required to automatically determine the ideal domain partition given an error bound. Translated into our context, we desire to determine the location of the minimum number of segments that approximate our set of points $\{\rho_i, z_i\}$ within some error tolerance. The optimal solution cannot, in general, be obtained in linear time. But we are interested in computing good approximations within this time bound.

We present two alternatives that solve this problem in linear time. The first one is *successive pivoting*, which is an intuitive approach that is easy to implement, and does not require knowledge of the sensor's inner workings. It is likely to fail, however, with very noisy measurements or with scanned surfaces that are not close to the sensor (noise dispersion increases with the distance to the sensor). The second approach is *segmentation with Chebyshev pre-smoothing*, which first transforms the data set into a more favorable coordinate system (a step that requires knowledge about the physics of the sensor), then fits the set with the weighted sum of the first n Chebyshev polynomials, and finally uses this continuous approximation as a noise-free representation of the surface being segmented. The Chebyshev smoothing approach is more robust to noise in the measurement process.

B.2.1 Successive Pivoting

One problematic characteristic of the data set $\{\rho_i, z_i\}$ is that both elements in each coordinate pair contain noise (see Appendix B.1.2). This implies that, although perfect sensing produces a sequence of points monotically increasing in z, the real measurement sequence is not always increasing in z nor uniformly spaced. The pivoting approach tries to circumvent this problem

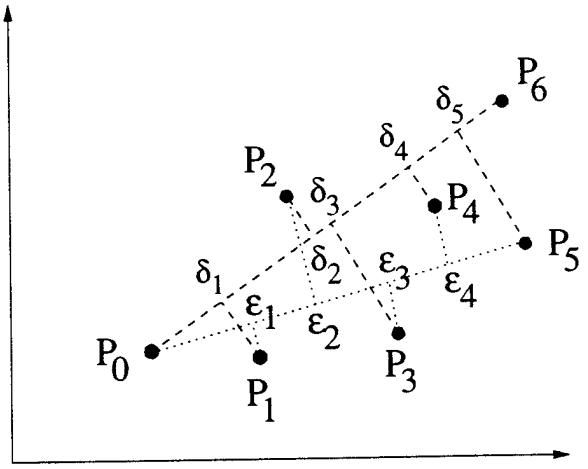


Figure B.5: The point-to-line distances change when the endpoint goes from p_k to p_{k+1}.

by exploiting the fact that the data was nevertheless extracted in a particular order, and by assuming that noise levels are such that the measurements $\{\rho_i, z_i\}$ are within a tolerable distance from the actual curve $\rho(z)$

Lets suppose that, in general, we have a sucession of points $(p_i)_{i=0}^m$ in $\Re^d$ space. These points are noisy measurements of a curve $\mathcal{C}$, and were acquired in an orderly fashion along the curve's arc. We select the point p_0 as the *pivot*, and define new segments successively by joining the pivot with points $p_1, p_2, \ldots$. At step k, we compute the squared distances from each point $(p_1, p_2, \ldots, p_{k-1})$ to the line defined by the pair (p_0, p_k), and average the quantities to obtain an error measure ξ_k:

$$\xi_k = \frac{1}{k-1} \sum_{i=1}^{k} d_i^2(p_i; \bar{p}_{0,k}), \tag{B.27}$$

where $\bar{p}_{0,k}$ is the segment defined by the pair (p_0, p_k), and $d_i^2(p_i; \bar{p}_{0,k})$ is the squared distance from point p_i to the line defined by the pair (p_0, p_k). If the computed error is above a threshold, we store $\bar{p}_{0,k}$ as an identified segment, and repeat the procedure with p_k as the new pivot. We repeat the algorithm until there are no more points to process.

In order to extract the segments in $O(d^c m)$ time, we need ways of computing the error ξ_{k+1} in $O(d^c)$ operations given the information used to compute ξ_k (see Figure B.5). With some vector algebra we can show that this is possible.

Let $\langle p_k \rangle_\alpha$ be the αth component of the vector $p_k \in \Re^d$. Assume the pivot is p_0, and define $\delta_i^2(p_k)$ as the distance from point p_i to the line $\bar{p}_{0,k}$. This quantity is expressed in vector

notation as:

$$\delta_i^2(p_k) = \mid p_i - p_0 \mid \cdot \left[\frac{(p_i - p_0) \cdot (p_k - p_0)}{\mid p_k - p_0 \mid} \right]^2, \tag{B.28}$$

$$= \sum_{\alpha=1}^{d} \langle p_i - p_0 \rangle_\alpha^2 - \left[\sum_{\alpha=1}^{d} \frac{\langle p_i - p_0 \rangle_\alpha \langle p_k - p_0 \rangle_\alpha}{\mid p_k - p_0 \mid} \right]^2. \tag{B.29}$$

Let $c_\alpha = \langle p_k - p_0 \rangle_\alpha / \mid p_k - p_0 \mid$, then

$$\delta_i^2(p_k) = \sum_{\alpha=1}^{d} [1 - c_\alpha^2] \langle p_i - p_0 \rangle_\alpha^2 - 2 \sum_{\alpha=2}^{d} \sum_{\beta=1}^{\alpha-1} c_\alpha c_\beta \langle p_i - p_0 \rangle_\alpha \langle p_i - p_0 \rangle_\beta. \tag{B.30}$$

Define the accumulators

$$S_{\alpha,\beta}(k-1) = \frac{1}{k-1} \sum_{i=1}^{k-1} \langle p_i - p_0 \rangle_\alpha \langle p_i - p_0 \rangle_\beta, \text{ then} \tag{B.31}$$

$$\xi_k = \sum_{\alpha=1}^{d} [1 - c_\alpha^2] S_{\alpha,\alpha}(k-1) - 2 \sum_{\alpha=2}^{d} \sum_{\beta=1}^{\alpha-1} c_\alpha c_\beta S_{\alpha,\beta}(k-1). \tag{B.32}$$

Hence, given the computation fo ξ_k, we can compute ξ_{k+1} by recomputing the constants $c_1, c_2, \ldots, c_d$ (which can be done in $O(d)$ operations), and by updating the $d(d-1)/2$ accumulators according to

$$S_{\alpha,\beta}(k) = \frac{k-1}{k} S_{\alpha,\beta}(k-1) + \frac{1}{k} \langle p_k - p_0 \rangle_\alpha \langle p_k - p_0 \rangle_\beta, \tag{B.33}$$
$$\forall\ \alpha = 1, 2, \ldots, d, \text{ and } \beta \geq \alpha,$$

which leads to the computation of each successive error in $O(d^2)$ operations.

Algorithm for Successive Pivoting

Overall, the pivoting algorithm requires $O(d^2 m)$ steps, meaning the process is linear in the number of data points. In our application, segmenting the curve $\rho(z)$ is always a problem in two dimensions, so the algorithm can be assumed to be $O(m)$. The pivoting procedure is summarized as follows:

Algorithm *Fit a chain of segments to a list of points*
Input:
a list of measurement points $\mathrm{L} = (p)_{i=0}^{m}$,
an error tolerance τ
Output: a list of segments $\mathcal{S} = (s)_j$

1. Select the first element in list $\mathcal{L}$ as the *pivot*. Let $k = 1$.
2. Define the line $\bar{p}_{0,k}$. Compute the error ξ_k with equation (B.32), with $c_\alpha = \langle p_k - p_0 \rangle_\alpha / \mid p_k - p_0 \mid$, and the accumulators updated according to equation (B.33). If $\xi_k < \tau$ increase k by one and repeat the step, else add the segment (p_0, p_k) to the list $\mathcal{S}$ and continue.
3. Remove all elements up to $k-1$ from list $\mathcal{L}$. If the list is not empty repeat the algorithm from step one, else the algorithm has reached termination.

B.2.2 Segmentation with Chebyshev Pre-Smoothing

As explained earlier, both elements in each coordinate pair (ρ_i, z_i) contains noise, and the sequence is not uniformly increasing in either of them. We could expect from the sensor geometry that a polar coordinate selection (r_i, θ_i) will result in a sequence of points of monotonically increasing and noise-free θ. This is approximately true, which means is false in general. The correct approach is to select coordinates ζ and ν defined as (see Appendix B.1.2):

$$\nu = \frac{\rho - \rho_o}{\rho - \rho_o + h}, \text{ and} \tag{B.34}$$

$$\zeta = \frac{z}{\rho - \rho_o + h}, \tag{B.35}$$

where ρ_o is the sensor calibration offset and h is a model parameter (see Appendix B.1.1). This coordinate pair is associated to pixel positions (x_p, y_p) of the sensor CCD camera by

$$\nu = d(x_p - c_x), \text{ and} \tag{B.36}$$

$$\zeta = b(y_p - c_y). \tag{B.37}$$

The main uncertainty present in the sensing process is during the detection of the pixel with highest intensity value for each row in the image. That is, for row y_p the sensor returns a peak position at $\hat{x}_p(y_p) = \text{round}[x_p(y_p) + \omega]$, where $x_p(y_p)$ is the true column location of the peak at row y_p, and ω is a random variable of mean zero and variance σ^2. Element x_p is the only noisy component of the pixel pair, hence ζ is noise free. Furthermore, for successive scanlines the value of y_p changes unarily, hence ζ is uniformly increasing.

The objective now is to fit an nth degree polynomial to the data sequence $(\zeta_i, \nu_i)_{i=1}^m$. The resulting polynomial $P_n(\nu)$ is intended to approximate the continuous curve $\nu(\zeta)$. A very good approximation can be obtained in $O(nm)$ time by using orthogonal polynomial representations. Posteriorly, this continuous approximation is used as a noise-free representation of the surface being segmented.

Chebyshev Polynomials

The Chebyshev polynomial of degree n is defined as $T_n(x) = \cos(n \arccos x)$[1]. $T_0(x) = 1$ and $T_1(x) = x$, and the polynomials of higher degree can be computed using the recursion $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. The polynomial $T_n(x)$ has n zeros in the interval $[-1, 1]$, located at $x = \cos(\frac{2k-1}{2n}\pi)$ for $k = 1, \ldots, n$.

Chebyshev polynomials are orthogonal in the interval $[-1, 1]$ under a weight $\sqrt{1-x^2}$:

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0 & i \neq j \\ \pi/2 & i = j \neq 0 \\ \pi & i = j = 0 \end{cases} . \tag{B.38}$$

Chebyshev polynomials satisfy a *discrete* orthogonal relation as well. If $x_k (k = 1, \ldots, m)$ are the m zeros of $T_m(x)$, then for $i, j < m$

$$\sum_{k=1}^{m} T_i(x_k)T_j(x_k) = \begin{cases} 0 & i \neq j \\ m/2 & i = j \neq 0 \\ m & i = j = 0 \end{cases} . \tag{B.39}$$

Given an arbitrary function $y = f(x)$ in the interval $[-1, 1]$, and the $x_k (k = 1, \ldots, m)$ zeros of $T_m(x)$, then

$$\varepsilon = \frac{1}{2} \sum_{k=1}^{m} (f(x_k) - P_n(x_k))^2 \tag{B.40}$$

is minimized when

$$P_n(x) = \sum_{j=0}^{n} c_j T_j(x) - \frac{1}{2} c_0, \text{ with} \tag{B.41}$$

$$c_j = \frac{2}{m} \sum_{k=1}^{m} y_k T_j(x_k), \tag{B.42}$$

[1]This equation looks trigonometric, but in fact yields a polynomial.

for $n \leq m-1$. The value of ε is zero when $n+1=m$, which means the data (x_k, y_k) is *interpolated.* For values of $n+1<m$ the data is *fitted* with a regression that minimizes the sum of squared approximation errors at the x_k nodes. It is clear from the equations that the approximation is computed in $O(nm)$ operations.

One interesting property of function approximation with Chebyshev polynomials is the *economization* property. Given the approximation $P_n(x) = 0.5c_0 + c_1T_1(x) + \ldots + c_nT_n(x)$ that minimizes ε, the truncation $P_{n-1}(x) = 0.5c_0 + c_1T_1(x) + \ldots + c_{n-1}T_{n-1}(x)$ is the polynomial of degree $n-1$ that minimizes the same measure. Finally, the Chebyshev approximation is nearly equal to the *minimax* polynomial, which is the one that has the minimizes the maximum deviation from the true function $y = f(x)$.

Approximation of $\zeta(\nu)$

Given the succession $(\zeta_i, \nu_i)_{i=1}^m$, we want the polynomial of degree $n < m-1$ such that the sum of squared residuals $(P_n(\zeta_i) - \nu_i)^2$ is minimized. This polynomial, if expressed as the weighted sum of the first n Chebyshev funtions, can be computed in $O(nm)$ operations if the approximation nodes are the roots of the mth Chebyshev polynomial. In order to satisfy this last condition we are going to transform the uniformly spaced $(\zeta_i)_{i=1}^m$ into the m roots $(\chi_i)_{i=1}^m$ of $T_m(\chi)$.

Granted that the roots of $T_m(\chi)$ are given by $\chi_i = \cos(\frac{2i-1}{2m}\pi)$, the transformation

$$\chi_i = \cos\left(\left[\frac{\zeta_i - \zeta_1}{\zeta_m - \zeta_1} + \frac{1}{2m}\right]\pi\right), \ \forall i = 1, \ldots, m, \tag{B.43}$$

maps ζ_i into χ_i for every $i = 1, \ldots, m$. It is indeed possible to transform uniform approximation nodes into the ones generated by the roots of $T_m(\chi)$.

Notice that equation (B.42) does *not* requires the roots of $T_m(\chi)$ explicitly, only the values of the function evaluated at these points (which we have already). The existence of the transformation of $(\zeta_i)_{i=1}^m$ into $(\chi_i)_{i=1}^m$ is required for evaluation of the function at points between the approximation nodes.

Economization

The next step is the economization of the approximation $P_n(\chi)$. From equation (B.36), we see that if the peak detection process contains noise of variance σ^2, then ν will have standard deviation $\mid d \cdot \sigma \mid$. By the Chebyshev's theorem in probability theory, for any probability distribution, 75% of the samples will fall within a distance of two standard deviations. So a

reasonable economization criterion is to select the degree $l < n$ that mantains a proportion $0.75 < p < 1.0$ of the evaluations $P_l(\chi_i)$ within an error of $\mid 2d \cdot \sigma \mid$. One should *never* require the approximation to be within an error of less than $\mid d \cdot \sigma \mid$, as this is the highest possible precision given the randomness inherent to the sensing process; if we force the approximation below this error we will interpolate the data, thus failing to smooth out noise.

Segmentation of $P_l[\chi(\zeta)]$

Once we compute $P_l(\chi)$, we want to fit a chain of segments to the succession $(\zeta_i, P_l(\chi_i))_{i=1}^m$, where χ_i is the ith root of $P_m(\chi)$. The proposed procedure is as follows:

1. Let $j = 1$, and $k = 2$.
2. If $k = m$ terminate the procedure, else fit a line $y = m\zeta + c$ to $(\zeta_i, P_l(\chi_i))_{i=j}^k$ and compute
$$\xi_{j,k} = \max_{\zeta_j \leq \zeta \leq \zeta_k} \mid y - P_l[\chi(\zeta)] \mid .$$
3. If $\xi_{j,k} < \tau$ (where τ is some tolerance), then increment k by one and repeat the previous step. If not continue.
4. Store line (m, c) in list $\mathcal{S}$, and make $j = k$ and $k = k + 1$. Go back to step 2.

The idea behind the procedure is to successively fit a line to a set of points, and break the line whenever the approximation ceases to be a good one. The tolerance τ should be selected as a proportion of $\mid d \mid$, which is the minimum detectable change in ν given a resolution of one pixel (see equation (B.36)).

In order to execute the procedure in linear time we must fit a line to $(\zeta_i, P_l(\chi_i))_{i=j}^k$ in constant time given the calculations of the fit to $(\zeta_i, P_l(\chi_i))_{i=j}^{k-1}$. This is indeed possible by the use of accumulators. Let

$$S_{p,q}(j,k) = \sum_{i=j}^{k} \zeta_i^p P_l^q(\chi_i), \tag{B.44}$$

then the polynomial $y_n(\zeta) = c_{n+1}\zeta^n + \ldots + c_1$ that minimizes the sum of squared residuals $[y_n(\zeta_i) - P_l(\chi_i)]^2$ for $i = j, j+1, \ldots, k$ is given by

$$\begin{bmatrix} c_1 \\ \vdots \\ c_{n+1} \end{bmatrix} = \begin{bmatrix} k-j+1 & S_{1,0}(j,k) & \cdots & S_{n,0}(j,k) \\ S_{1,0}(j,k) & S_{2,0}(j,k) & \cdots & S_{n+1,0}(j,k) \\ \vdots & \vdots & \ddots & \vdots \\ S_{n,0}(j,k) & S_{n+1,0}(j,k) & \cdots & S_{2n,0}(j,k) \end{bmatrix}^{-1} \begin{bmatrix} S_{0,1}(j,k) \\ \vdots \\ S_{n,1}(j,k) \end{bmatrix}. \tag{B.45}$$

The optimal polynomial fit for $i = j, j+1, \ldots, k+1$ is the same as above, with the accumulators updated as follows:

$$S_{p,q}(j, k+1) \;=\; S_{p,q}(j,k) + \zeta_{k+1}^{p} P_l^q(\chi_{k+1}). \tag{B.46}$$

So for a line, these last set of equations imply that successive fits can be computed in $O(c)$ time given the previous one.

There is a problem when computing $\xi_{j,k}$, which in general takes $O(k-j+1)$ calculations. The way around this is to locally approximate the function $P_l[\chi(\zeta)]$ by fitting the last r points to a quadratic function $a_3\zeta^2 + a_2\zeta + a_1$. The approximation for the error is given in closed form by

$$\hat{\xi}_{j,k} \;=\; \min_{\zeta \in \{\zeta_{low}, \zeta^*, \zeta_{high}\}} \mid a_3\zeta^2 + (a_2 - m)\zeta + a_1 \mid, \text{ where} \tag{B.47}$$
$$\zeta_{low} = \zeta_{k-r+1}, \quad \zeta^* = \tfrac{m-a_2}{2a_3}, \quad \zeta_{high} = \zeta_k \;,$$

and m and c are the parameters of the line $y = m\zeta + c$ that best fit $(\zeta_i, P_l(\chi_i))_{i=j}^{k}$. We use estimate $\hat{\xi}_{j,k}$ to determine a new breaking point in the line construction. Equation (B.45) imply that the quadratic fit can also be updated in constant time. This approximation of $\xi_{i,k}$ is justified by the fact that if the current constructed line is kept close to the function $P_l(\xi)$ within interval $[j,k]$, then a local quadratic approximation over the reduced interval $[k-r+1, k]$ is even closer, meaning $\hat{\xi}_{j,k}$ is almost equal to the true value.

Algorithm for Segmentation with Chebyshev Pre-Smoothing

It has been shown by this point that our proposed segmentation procedure of $P_l[\chi(\zeta)]$ can be obtained with $O(m)$ operations. Given a selection of n, the function $P_n[\chi(\zeta)]$ (later economized into a polynomial of degree $l < n$) can be computed in $O(nm)$ time by using Chebyshev polynomials. Hence, the entire segmentation of a scan profile can be accomplished in $O(nm)$ operations, which is linear in the number of data points. Segmentation with Chebyshev pre-smoothing is summarized as follows:

Algorithm *Fit a chain of segments to a list of points*
Input:

a list of pixel points $\mathrm{L} = (x_p, y_p)_{i=0}^{m}$,
the sensor model parameters $\{b, d, h, c_x, c_y\}$ (see Appendix B.1.1),
a constant integer n -the maximum degree of the polynomial fit,
the variance σ^2 of the peak detection process in the sensor,
an economization proportion factor $0.75 < p < 1.0$,
the segmentation precision $\tau \sim \mid d \mid$,
a window size r -the scope of the local quadratic estimation

Output: a list of line paramters $\mathcal{S} = (m, c)_j$

1. Compute $(\zeta_i, \nu_i)_{i=1}^{m}$ from $(x_p, y_p)_{i=0}^{m}$ according to equations (B.36) and (B.37). Compute the approximation nodes $\chi_i = \cos(\frac{2i-1}{2n}\pi)$ for $i = 1, \ldots, m$ (the zeros of the mth Chebyshev polynomial).
2. Compute the polynomial approximation $\nu = P_n[\chi(\zeta)]$ with equations (B.41) and (B.42). Use transformation (B.43) if internodal evaluations of the function are required.
3. Compute the economization $P_l[\chi(\zeta)]$ by selecting the minimum value of l that keeps a proportion p of the evaluations $P_l(\chi_i)$ within an error of $\mid 2d \cdot \sigma \mid$.
4. Initialize $j = 1$, and $k = 2$.
5. If $k = m$ terminate the algorithm, else fit a line $y = m\zeta + c$ to $(\zeta_i, P_l(\chi_i))_{i=j}^{k}$ using equations (B.45) and (B.46) to compute the linear fit and the local quadratic approximation of scope r. Use equation (B.47) to approximate $\xi_{j,k}$. If $\xi_{j,k} < \tau$ then increment k by one and repeat the previous step. If not continue.
6. Store line (m, c) in list $\mathcal{S}$, and make $j = k$ and $k = k + 1$. Go back to step 5.

Appendix C

Pattern-Tracking in an Image Sequence

This appendix presents a method to track moving non-rigid objects, with emphasis primarily placed on the vision aspects. The algorithm was implemented and some basic experiments were performed with a stationary camera. It is hoped that these techniques can be further developed and eventually incorporated into the autonomous observer.

The method is based on a comparison between a model and an image, and the comparison function employed is the partial Hausdorff distance. The model and the image are bitmaps extracted from a sequence of gray-level images. The target's motion in the image is decomposed into two parts: a two-dimensional motion, corresponding to the change in the target's position in the image space, and a two-dimensional shape change, corresponding to a new aspect (viewing direction) of the target.

C.1 Introduction

The target tracking problem has received a good deal of attention in the computer vision community over the last years. Several techniques have been reported in the literature, and a variety of features have been proposed to perform the tracking.

Some methods attempt to track 3-D objects by using multiple views [28]. Others use contours as features to perform the tracking [8, 27, 66]. Some contour methods use curve characterization [64] and active contours [20] to handle object deformations. Other approaches use either color information [79] to facilitate the tracking, or a combination of regions computed by color segmentation and contours [78]. Finally, some techniques identify motions by detecting

variations of some global characteristic of the image such as texture or statistical invariants [67].

We developed a method based on a comparison between a model and image. The model and the image are bitmaps extracted from a sequence of gray levels images using a edge detector similar to [15]. The input data are handled as an array of binary elements, where the characteristic pixels (nonzero points) belonging to edges in the model and image are not required to be concatenated. The partial Hausdorff distance is used in order to measure the resemblance of the image with the model [36].

C.2 The Target Tracking Method

This appendix describes a method to track moving non-rigid objects. The method is based on the assumption that the motion of a non-rigid object in three dimensional space can be characterized using a two-dimensional representation.

The target's motion in a two-dimensional representation (image) can be decomposed into two parts:

- A two-dimensional motion in the image, corresponding to the change in the target's position in the image space.
- A two-dimensional shape change, corresponding to a new aspect of the target.

The tracking is done using a comparison between an image and a model. It is possible to represent the model and the image with two binary arrays, where the (i, j) element in each array is 1 if it corresponds to an edge, or 0 otherwise. Figure C.1a shows the original gray level image, and Figures C.1b and C.1c show the edges of both the image and model.

To reduce the number of operations, and consequently the time of computation, it is necessary only to search the model in a region contiguous with the target position in the previous frame of the sequence. This assumes a bound on the target's speed that guarantees that the target's new position lies inside the region of examination. We also assume a known initial target position, and that the shape of an object does not change significantly between two successive frames of an image sequence.

C.3 Comparing the Image with the Model

To measure the resemblance of an image with the model we use the partial Hausdorff distance.

a *b* *c*

Figure C.1: (*a*) Original gray-level image. (*b*) Edges of original image. (*c*) Edges of model.

Given two sets of points P and Q, the Hausdorff distance is defined as (see [36])

$$H(P,Q) = \max(h(P,Q), h(Q,P))$$

where

$$h(P,Q) = \max_{p \in P} \min_{q \in Q} \| p - q \| \tag{C.1}$$

and $\| \cdot \|$ is some norm for measuring the distance between two points p and q. The function $h(P,Q)$ (distance from set P to Q) is a measure of the degree in which each point in P is near to some point in Q. A small value of $h(P,Q)$ implies that every point in P is close to a point in Q. The Hausdorff distance is the maximum among $h(P,Q)$ and $h(Q,P)$. Thus the Hausdorff distance measures the degree to which each point of P is near some point Q and vice versa.

By computing the Hausdorff distance in this way we obtain the most mismatched point between the two compared shapes; consequently, it is very sensitive to the presence of any outlying points. For that reason it is often appropriate to use a more general rank order measure, which replaces the maximization operation with a rank operation. This measure (partial distance) is defined as:

$$h_k = K^{th}_{p \in P} \min_{q \in Q} \| p - q \|, \tag{C.2}$$

where $K^{th}_{p \in P} f(p)$ denotes the K^{-th} ranked value of $f(p)$ over the set P. That is, if we consider the points in P to be in sequence ordered by their values $f(p_1) \leq \ldots \leq f(p_n)$, the K^{-th} element in this sequence, $f(p_K)$, is the K^{-th} ranked value. For example, the n_{-th} ranked value is the maximum (the largest element in the sequence), and the $n/2{-}th$ ranked value is the median.

One interesting property of the Hausdorff distance and the "partial distance" is the asymmetry inherent in the computation. The fact that every point of P (or subset of P) is near some point of Q says nothing about whether every point of Q (or subset of Q) is near some point of P. In other words, $h_{k1}(P,Q)$ and $h_{k2}(Q,P)$ can attain very different values. In fact each one of the two values give different information.

The term $h_{k1}(P,Q)$ is the unidirectional partial distance from the model to the image, and $h_{k2}(Q,P)$ the unidirectional partial distance from the image to the model, where $P = M_t$ is the model and $Q = I_t$ is the image or region of the image given in at t time of one sequence. The maximum of these two values defines the partial Hausdorff distance.

C.4 Finding the Model

The first task is to define the position of the model M_t in the next image I_{t+1} of the sequence. The search of the model in the image (or image's region) is done in some selected direction.

The minimum value of $h_{k1}(M_t, I_{t+1})$ identifies the best "position" of M_t in I_{t+1}, under the action of some group of translations G. It is possible also to identify the set of translations of M_t such that $h_{k1}(M_t, I_{t+1})$ is no larger than some value τ; in this case there may be multiple translations that have essentially the same quality. However, rather than computing the single translation that gives the minimum distance or the set of translations, such that its corresponding h_{k1} is no larger than τ, it is possible to find the first translation, such that its associated h_{k1} is no larger than τ, for one search direction given.

Although the first translation which the associated $h_{k1}(M_t, I_{t+1})$ is less than τ, it is not necessarily the best one. When τ is small, the translation should be quite good. Besides in the updating of the model (in order to renew the shape's change of the model), it is also possible to correct its position, thus reducing the induced error. This is better then computing the set of all valuable translations; hence, the computing time is significantly less.

In order to compute the term

$$\min_{q \in Q} \| p - q \|$$

of (C.2), it is necessary to compute the distance from any position i, j in the image Q to the nearest nonzero point in this image. As the model is superimposed on the image, it computes the minimal distance between the model points (nonzero pixels) to the points in the image (also nonzero pixels). This distance has been referred to as a distance transform, because it gives the distance from any point p to the nearest point in a set of source points Q. It has also been called the Voronoi distance of Q by analogy to Voronoi diagrams, which

specify the locations equidistant from two or more points of a given set.

C.4.1 The Voronoi Distance $D_q(i,j)$

There are many methods of computing the Voronoi distance or an approximation of it [9], [19]. We use an algorithm similar to [59], which produces distance transform values that are exact up to machine precision with respect to some norm of the distance.

This algorithm first processes each row independently. For each row of $Q(i,j)$, it calculates the distance to the nearest pixel in the row; for each (i,j) it finds Δi such that $Q(i+\Delta i, j) \neq 0$ or $Q(i-\Delta i, j) \neq 0$ and that Δi is the minimum nonnegative value for which this is true. It then computes the distances to the nearest pixel in each column in the same mode.

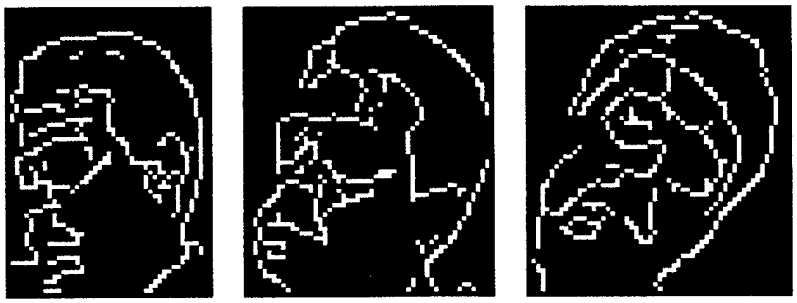
To reduce the number of operations, the distance to the nearest point for a given position (i,j) is computed by comparing the distances of the possibles point candidates, just in the zone determined for the minimum value between the distance to the nearest pixel in the row, and the distance to the nearest pixel in the column. A look-up table that depends on the distance-norm is used to compute the value of the distances.

C.4.2 Two Techniques that Decrease the Computation Time of the Partial Hausdorff Distance

The computation of the partial Hausdorff distance as a function of translation can take a significant amount of time to run because it considers every possible translation of the model until it is finds one for which $h_k < \tau$.

The first technique consists of computing the number of points m of the model, which are at distance $h_k > \tau$ for a given translation; n is the total number of points in the model. If $m > n - k_{-th}$, it is not necessary to continue scanning the model, in view of the fact that it is not possible to have k_{-th} to a distance $h_k < \tau$; consequently, we can finish the analysis for that translation.

One second technique relies on the order in which the space of possible translations are scanned. Assume that the order is a row at a time in the increasing i direction, to superimpose the model on the image. Let $D'_{q(+i)}(i,j)$ be the distance in the increasing direction i to the nearest location where the Voronoi distance is $D_q(i,j) < \tau$. We can use $D'_{q(+i)}(i,j)$ to determinate how far we would have to move in the increasing i direction to find a place where $D_q(i,j) < \tau$. The model will translate at this location when $m > n - k_{-th}$ (see [37] for more details).



a *b* *c*

Figure C.2: Evolution of model updates.

C.5 Updating the Model

Having found the first translation g such that $h_{k1} < \tau$, we now have to build the new model M_{t+1} by determining which pixels of the image I_{t+1} are part of this new model. Figures C.2a, C.2b, and C.2c show different model updates.

The model is updated by using the distance $h_{k2}(I_{t+1}, g(M_t))$ as a criterion for selecting the subset of images points I_{t+1} that belong to M_{t+1}. The new model is defined by:

$$M_{t+1} = \{q \in I_{t=1} \mid h_{k2}(I_{t+1}, g(M_t)) < \delta\}, \tag{C.3}$$

where $g(M_t)$ is the model in the time t under the action of the translation g, and δ controls the degree to which the method is able to track objects that change shape. In practice this is done by dilating $g(M_t)$ by an amount δ in each direction (columns and rows). $\delta = 0$ will cause the tracker to lose an object after several frames, even if the object does not actually change due to noise; on the contrary δ too large could 'tracking' the background.

To allow for models that might be changing in size, this size is increased whenever there are a significant number of nonzero pixels near the boundary, and is decreased in the contrary case. The model's position is improved according to the position where the model's boundary was defined.

The initial model must be computed with a different method because there is no previous model. The user specifies a rectangle in the frame that contains the model (see Figure C.3), and one image of the background is taken without the model.

All of the points belong to the background image, and those not in the model are eliminated to obtain a initial model without noise. It is assumed that the camera does not move between

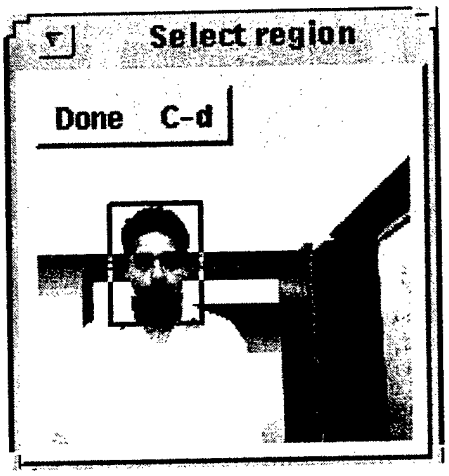


Figure C.3: Initial model

the two images.

With this initial model the tracking is begun, iteratively finding the new position of the target and updating the model. The tracking of the model is successful if

$$k1 > fM \mid h_{k1}(M_t, I_{t+1}) < \tau \tag{C.4}$$

and

$$k2 > fI \mid h_{k2}(I_{t+1}, g(M_t)) < \delta \ , \tag{C.5}$$

in which fM is a fraction of the number total of points of the model M_t and fI is a fraction of image's point of I_{t+1} superimposed to $g(M_t)$.

C.6 Extensions over the General Method

The target tracking method presented in this appendix is based in the one introduced in [37] and [36]. This section enumerates some of the extensions over the general method.

- Only a small region of the image is examined to obtain the new target position, as opposed to the entire image. In this way the computation time is decreased significantly. The idea behind a local exploration of the image is that if the execution of the code is quick enough, the new target position will then lie within a vicinity of the previous one. We are trading the capacity to find the target in the whole image in order to increase the speed of computation of the new position and shape of the model. In this way the

robustness of the method is increased to handle target deformations, since it is less likely that the shape of the model changes significantly in a small δt. In addition, this technique allows the program to report the target's location to any external systems with a higher frequency (for an application see [6]).

- Instead of computing the set of translations of M_t, such that $h_{k1}(M_t, I_{t+1})$ is no larger than some value τ, we are finding simply the first translation whose $h_{k1}(M_t, I_{t+1})$ is less than τ. This strategy decreases the computational time. However, this technique could cause an accumulation of error that induces a tracking drift in a direction opposite to the search. It is possible to avoid drifting by resizing the target bounding box once its new position is computed, and by reducing the tolerance of the model and image mismatch.

- In order to compute the Voronoi distance $D_q(i, j)$, we use an algorithm which produces distance transform values that are exact up to machine precision. The algorithm structure is the same for any distance metric $D[(x_1, y_1), (x_2, y_2)]$ that is a function of the relative values of the pairs (x_1, x_2) and (y_1, y_2), is nondecreasing in $|x_1 - x_2|$, and nondecreasing in $|y_1 - y_2|$ (e.g., city block, chess-board and Euclidean distances).

C.7 Conclusion and Future Work

In this appendix we have described a target tracking method that basically consists of a comparison between an image and a model. The comparison function employed is the partial Hausdorff distance. Several techniques that decrease computation time were presented.

The basic ideas of our implementation are the following:

- The image of a solid object moving in space can be decomposed into two components: two-dimensional motion and a two-dimensional deformation.

- The search of the target is done only in a vicinity of the image near to the previous target location, instead of the entire space image.

- We find the first translation for which unidirectional Hausdorff distance $h_{k1}(M_t, I_{t+1})$ is satisfied.

This method was implemented in C and C++, and the code is capable of processing a frame in about 0.3 seconds. Processing includes smoothing, edge detection, target localization, and model updating for a video image of (160×120 pixels) on a SPARC 20. The sequences in Figures C.4 show the evolution of the model for a series of gray-level images.

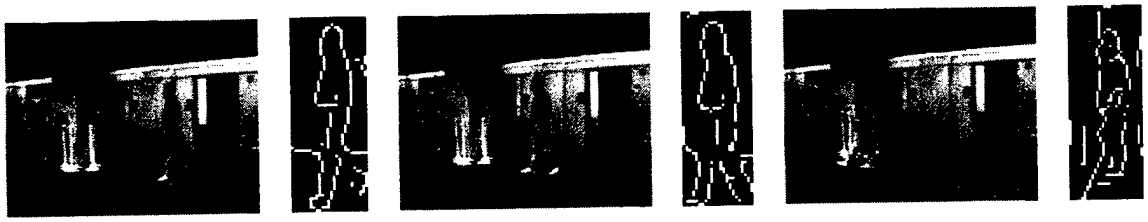


Figure C.4: Evolution of the model for a sequence of gray-level images.

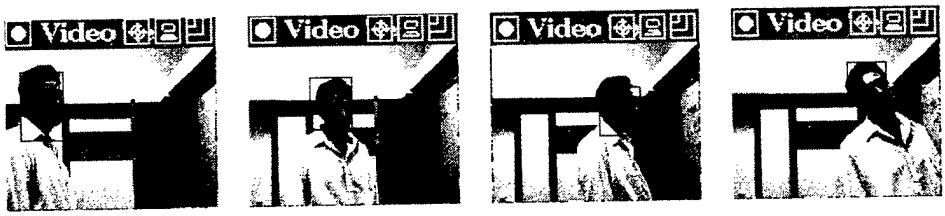


Figure C.5: Some intermediate results of a real-time target tracking experiment.

Figure C.5 shows the results of one of our tests. The sequence is composed by different snapshots of a target tracking experiment executed in real time. The images correspond to the tracking of a person's head, which is enclosed in each image with bounding box.

In terms of adding functionality to the system, there are several possible extensions:

First: we are extending the algorithm in order to handle different models (different views) of a single target. The objective is to improve robustness toward changes in shape, and to allow target recovery in case of loss.

Second: we plan to study image preprocessors that would enhance those image features that are appropriate to our method.

Third: currently the user specifies the location and size of the feature in the initial frame to specify the model to track. We would like to explore the possibility of an automatic feature identification system. This system should use several attributes of the image (such as color, texture, shape) to select the initial model.

Bibliography

[1] T. Başar and P. R. Kumar. On worst case design strategies. *Comput. Math. Applic.*, 13(1-3):239–245, 1987.

[2] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory.* Academic Press, London, 1982.

[3] J. E. Banta, Y. Zhien, X. Z. Wang, G. Zhang, M. T. Smith, and M. A. Abidi. A "best-next-view" algorithm for three-dimensional scene reconstruction using range images. In *Proc. SPIE vol. 2588*, pages 418–29, 1995.

[4] J. Barraquand and P. Ferbach. Motion planning with uncertainty: The information space approach. In *IEEE Int. Conf. Robotics & Automation*, pages 1341–1348, 1995.

[5] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. In G. Giralt and G. Hirzinger, editors, *Proc. of the 7th International Symposium on Robotics Research*, pages 249–264. Springer, New York, NY, 1996.

[6] C. Becker, H. González-Baños, J.-C. Latombe, and C. Tomasi. An intelligent observer. In *Proc. of International Symposium on Experimental Robotics*, pages 94–99, 1995.

[7] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[8] A. Blake and et Al. Affine-invariant contour tracking with automatic control of spatiotemporal scale. In *4th International Conference on Computer Vision*, pages 66–75. IEEE Computer Society Press, April 1993.

[9] G. Borgefors. Distance transforms in digital images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:344–371, 1986.

[10] A. J. Briggs and B. R. Donald. Robust geometric algorithms for sensor planning. In J.-P. Laumond and M. Overmars, editors, *Proc. 2nd Workshop on Algorithmic Foundations of Robotics*. A.K. Peters, Wellesley, MA, 1996. To appear.

[11] P. J. Burt, C. Yen, and X. Xu. Local correlation measures for motion analysis: a comparative study. In *Proc. IEEE Conf. Pattern Recognition and Image Processing*, pages 269–274, 1982.

[12] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. IEEE Conf. on Foundations of Computer Science*, pages 49–60, 1987.

[13] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[14] J. F. Canny. On computability of fine motion plans. In *IEEE Int. Conf. Robotics & Automation*, pages 177–182, 1989.

[15] J.F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[16] Z. Chen and C.M. Huang. Terrain exploration of a sensor-based robot moving among unknown obstacles of polygonal shape. In *Robotica*, volume 12, pages 33–44, 1994.

[17] C. I. Conolly. The determination of next best views. In *IEEE Int. Conf. on Robotics and Automation*, pages 432–435, 1985.

[18] M. Dai, P. Baylou, and M. Najim. An efficient algorithm for computation of shape moments from run-length codes or chain codes. *Pattern Recognition*, 25(10):1119–1128, 1992.

[19] P.E. Danielsson. Euclidean distance mapping. *Comput. Graphics Image Processing*, 14:227–248, 1980.

[20] P. Delagnes, J. Benois, and D. Barba. Adjustable polygons: a novel active contour model for objects tracking on complex background. *Journal on Communications*, 45:83–85, 1994.

[21] B. R. Donald. Planning multi-step error detection and recovery strategies. *Int. J. Robotics Research*, 9(1):3–60, 1990.

[22] M. Erdmann. Randomization in robot tasks. *Int. J. Robotics Research*, 11(5):399–436, October 1992.

[23] M. Erdmann. Randomization for robot tasks: Using dynamic programming in the space of knowledge states. *Algorithmica*, 10:248–291, 1993.

[24] M. A. Erdmann. On motion planning with uncertainty. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, August 1984.

[25] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Trans. Robotics & Automation*, 8(3):313–326, June 1992.

[26] M.A. Garcia, A.D. Sappa, and L. Basanez. Fast generation of adaptive quadrilateral meshes from range images. In *IEEE Int. Conf. on Robotics and Automation*, 1997.

[27] G. Garibotto. Motion tracking of connected edge contours. In *Time-varying Image Processing and Moving Object Recognition. Proc. 3rd Int. Workshop*, pages 323–330, 1990.

[28] D.M. Gavrila and et Al. 3-d model-based tracking of humans in action: a multi-view approach. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, pages 73–80, 1996.

[29] L. Guibas and J. Stolfe. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *AMC Trans. Graphics*, 4(2):74–123, 1985.

[30] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. 1st Workshop on Algorithmic Foundations of Robotics*, pages 269–282. A.K. Peters, Wellesley, MA, 1995.

[31] L.J. Guibas, J.C. Latombe, S.M. Lavalle, D. Lin, and R.Motwani. Visibility-based pursuit-evasion problem. In *Int. J. of Computational Geometry and Applications*, Special Issue on the CGC Workshop on Comp. Geom., 1997. Invited paper.

[32] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 1987.

[33] M.K. Hu. Visual pattern recognition by moment invariants. *IRE Trans. on Information Theory*, 1962.

[34] S. Hutchinson. Exploiting visual constraints in robot motion planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1722–1727, 1991.

[35] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Trans. Robotcis & Automation*, 12(5):651–670, October 1996.

[36] D.P. Huttenlocher and et Al. Comparing images using the hausdorff distance. *Trans. on Pattern Analysis and Machine Intelligence*, 15(9):850–863, Sept 1993.

[37] D.P. Huttenlocher, J.J Noh, and W.J. Rucklidge. Tracking non-rigid objects in complex scenes. In *Proc. 4th Int. Conf. on Computer Vision*, pages 93–101, 1993.

[38] R. Isaacs. *Differential Games.* Wiley, New York, NY, 1965.

[39] K. Kakusho, T. Kitahashi, K. Kondo, and J.-C. Latombe. Continuous purposive sensing and motion for 2d map building. In *Proc. IEEE Int. Conf. Systems, Man, & Cybernetics*, pages 1472–1477, 1995.

[40] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *Int. J. Robotics Research*, 5(3):72–89, 1986.

[41] L. E. Kavraki. Computation of configuration-space obstacles using the Fast Fourier Transform. *IEEE Trans. Robotics & Automation*, 11(3):408–413, 1995.

[42] J.M. Kleinberg. On-line search in a simple polygon. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 8–15, 1994.

[43] D.J. Kriegman, E. Triendl, and T.O. Binford. Stereo vision and navigation in buildings for mobile robots. *IEEE Trans. on Robotics and Automation*, 5(6):1722–1727, 1991.

[44] P. R. Kumar and P. Varaiya. *Stochastic Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[45] R. E. Larson and J. L. Casti. *Principles of Dynamic Programming, Part II.* Dekker, New York, NY, 1982.

[46] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[47] S. M. LaValle. *A Game-Theoretic Framework for Robot Motion Planning.* PhD thesis, University of Illinois, Urbana, IL, July 1995.

[48] S. Lavallée, J. Troccaz, L. Gaborit, A. L. Benabid P. Cinquin, and D. Hoffman. Image-guided operating robot: A clinical application in stereotactic neurosurgery. In R. H. Taylor, S. Lavallée, G. C. Burdea, and R. Mösges, editors, *Computer-Integrated Surgery*, pages 343–351. Mit Press, Cambridge, MA, 1996.

[49] A. Lazanas and J.C. Latombe. Landmark-based robot navigation. *Algorithmica*, 13:472–501, 1995.

[50] A. Lazanas and J.C. Latombe. Motion planning with uncertainty: A landmark approach. *Artificial Intelligence*, 76(1-2):287–317, July 1995.

[51] T.S. Levitt, D.T. Lawton, D.M. Chelberg, and P.C. Nelson. Qualitative navigation. In *Proc. DARPA Image Understanding Workshop*, pages 447–465, 1987.

[52] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[53] L. Matthies, T. Kanade, and R. Szeliski. Kalman filter-based algorithms for estimating depth from image sequences. *Int. Journal of Computer Vision*, 3:209–238, 1989.

[54] J. Maver and R. Bajcsy. Occlusions as a guide for planning the next view. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(5):417–433, May 1993.

[55] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *Journal of ACM*, 35(1):18–44, January 1988.

[56] V.S. Nalwa. *A Guided Tour of Computer Vision.* Addison Wesley, 1993.

[57] B. K. Natarajan. The complexity of fine motion planning. *Int. J. Robotics Research*, 7(2):36–42, 1988.

[58] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, NY, 1987.

[59] D.W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP Graphical Models and Image processing*, 54(1):56–74, Jan 1992.

[60] N. P. Papanikolopoulos, P. K. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision. *IEEE Trans. Robotics & Automation*, 9(1):14–35, February 1993.

[61] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alani and D. R. Lick, editors, *Theory and Applcation of Graphs*, pages 426–441. Springer-Verlag, Berlin, 1976.

[62] R. Pito. A solution to the next best view problem for automated cad model acquisition of free-form objects using range cameras. Technical Report 95-23, GRASP Lab, University of Pennsylvania, May 1995.

[63] W.H. Press, S.A. Teukolsky, W.T. Vettering, and B.P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1994.

[64] J.M. Rehg and T. Kanade. Visual tracking of high dof articulated structures: an application to human hand tracking. In *Proc. 3rd European Conf. on Computer Vision*, volume 2, pages 35–46, 1994.

[65] T.H. Reiss. The revised fundamental theorem of moment invariants. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(8), 1991.

[66] D. Sage and et Al. A contour feature tracking system. In *Proc. of the 7th Int. Conf. on Robot Vision and Sensory Controls: ROVISEC-7 Advanced Sensor Technology*, pages 247–256. IFS (Publications), 1988.

[67] J. Shi and C. Tomasi. Good features to track. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[68] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Comput.*, 21(5):863–888, October 1992.

[69] H. Takeda, C. Facchinetti, and J.C. Latombe. Planning the motions of a mobile robot in a sensory uncertainty field. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(10):1002–1017, Oct 1994.

[70] R. Talluri and J. K. Aggarwal. Mobile robot self-location using model-image feature correspondence. *IEEE Trans. Robotics & Automation*, 12(1):63–77, February 1996.

[71] R. Y. Tasi and T. S. Huang. Uniqueness and estimation of three-dimensional motion parameters of rigid objects with curved surfaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-6(1):13–27, 1984.

[72] Q. Tian and M. N. Huhns. Algorithms for subpixel registration. *Computer Vision Graphics and Image Processing*, 35:220–233, 1986.

[73] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proc. ACM SIGGRAPH*, pages 311–318, 1994.

[74] K. Umeda and K. Ikushima. 3d shape recognition by distributed sensing of range images and intensity images. In *IEEE Int. Conf. on Robotics and Automation*, 1997.

[75] J. Weng, T. S. Huang, and N. Ahuja. Motion and structure from two perspective views: Algorithms, error analysis, and error estimation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(5):451–476, 1989.

[76] L. Wixson. Viewpoint selection for visual search. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 800–805, 1994.

[77] G. A. Wood. Realities of automatic correlation problems. *Photogram. Eng. and Rem. Sens.*, 49:537–538, 1983.

[78] C. Wren and et Al. Pfinder: Real-time tracking of the human body. In *Proc. SPIE Int. Society for Optical Engineering*, volume 2615, pages 89–98, 1996.

[79] Du Yue and et Al. A color projection for fast generic target tracking. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 73–80, 1995.