Mapping and Sensor Fusion for an Autonomous Vehicle

by

Steve J. Steiner

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science

and

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author .....
Department of Electrical Engineering and Computer Science
May 27, 1996

Certified by ...
avid S. Kang
ect Manager
's Supervisor

Certified by .....
Gill Pratt
Professor
esis Supervisor

Accepted by .....
Morganthaler
Chairman, Departmental Committee on Graduate Theses

Mapping and Sensor Fusion for an Autonomous Vehicle

by
Steve J. Steiner

Submitted to the Department of Electrical Engineering and Computer Science
on May 27, 1996, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science
and
Master of Engineering in Computer Science and Engineering

Abstract

The problem of mapping and sensor fusion for an autonomous vehicle is explored. A novel data structure for recording the sensor readings during operation is presented. The data structure can accommodate a range of possible fusion strategies, including dynamically switching strategies during operation. Of particular interest is the hierarchical structure and the use of delayed evaluation. The structure allows for both global and local mapping to occur in one map. It makes possible the dynamic growth of the map and provides a speed and memory improvement versus a standard matrix. The concept of delayed evaluation, which allows for multiple fusion strategies to be implemented on the same data, is presented. A trial implementation on the autonomous robot, Companion, provides a test of the feasibility of the data structure.

Thesis Supervisor: David S. Kang
Title: Project Manager

Thesis Supervisor: Gill Pratt
Title: Professor

Acknowledgments

I would like to thank for parents for believing in me and sending me through MIT. I also thank David Kang for giving me the opportunity to work in the Unmanned Vehicle lab. The lab has provided me with a number of valuable experiences. I also must thank Bob Powers for a large amount of invaluable feedback on the content of this thesis, and on my work in general.

As for the students in the lab, I would like to thank Sean Adam for a large number of hardware debugging sessions, and Bill Kaliardos for fixing a large number of mechanical difficulties a few of which we did cause. Chuck Tung should to be immortalized for his work on the gyro, and his general skill in everything electrical. Thanks for coming through in a crunch guys.

Finally my partner in this endeavor, Terence Chow, deserves a great deal of credit. He is the real spirit behind the Companion project. Thanks for cracking the whip Terrence.

This thesis was supported by the Charles Stark Draper Laboratory, Inc. Publication of this thesis does not constitute approval by The Charles Stark Draper Laboratory, Inc., of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Steve Steiner

Contents

List of Figures

Chapter 1

Introduction

This thesis involves the design, construction and evaluation of a system for analyzing real-time sensor readings and creating a world model for an autonomous mobile robot. The project is primarily an engineering task. The Companion mobile robot is being developed at the Unmanned Vehicle Lab at Charles Stark Draper Laboratory. The Companion robot uses the *mapper* program created for this thesis in close conjunction with the path planning and control system created by Terence Chow, another student at the lab. These systems in conjunction allow the robot to achieve a high degree of autonomy in unknown, largely unstructured environments.

The introduction will provide background information on the robot platform, an analysis of the engineering task to be solved, and an overview of the methods which were considered in solving those problems. Where it is reasonable to do so the rationale for engineering decisions will be made explicit.

1.1 Overview

1.1.1 Certainty Grids

There are few practical methods for creating maps for autonomous vehicles. One obvious method attempts to accurately locate the boundaries of obstacles. Successful implementations of this method often need to make assumptions about what types

of obstacles will be encountered. Moravec [7] and Elfes [4] had early success with an alternate method which models space as a grid with each cell determining the probability that it is occupied. This method does not require one to make assumptions about the types of obstacles that are encountered.

A certainty grid partitions space into a grid. Each cell in the grid contains a number which represents the certainty that the area is occupied. All cells in the grid will typically start in an uncertain state. As readings are obtained, the cells are updated by adjusting the certainties. For example, a sonar return will cause the cells which fall inside the range cone to be updated toward an unoccupied state. The cells which fall along the edge of the range will be updated toward an occupied state. The certainty numbers provide a range of values which allows one to use a search program which plans paths along the areas that are more certain of being unoccupied.

The certainty grid began as a method for integrating multiple sonar readings in the Dolphin sonar mapping and navigation system [4]. The Dolphin architecture merges multiple views, or maps created from single positions into local maps. It further merges multiple local maps into a global map. In the original formulation each map is kept as two separate grids, one for empty areas and one for occupied areas. These grids are merged to provide a threshold value. Stewart [8] found that the two maps were redundant. A single map suffices without information loss, and provides a performance improvement. The use of certainty values was replaced with probabilities and the use of Bayesian update rules [7]. This improvement gave the certainty grid method a firmer theoretical basis.

Moravec [7] identifies the use of certainty grids as one of the keys to creating a robust control system. Earlier attempts based on locating obstacle boundaries had difficulty dealing with sensor errors and conflicting readings. Certainty grids provide a simple, understandable method of gaining a view of the world robust against errors and conflicting readings. Certainty grids were also used in solutions to many other problems which are not as clearly amenable to a grid solution. It is not clear that converting a certainty grid to a polygon representation is the best method of creating a robust polygon map. The algorithm to do the conversion does not run in real time

and the conversion can be suspect for a number of reasons, including the possibility that the grid is not fine enough. The use of certainty grids to solve these problems likely stems from the success in providing robust mapping capabilities.

Another approach which also uses a grid, but has better real time performance for robots with fast active sensors is the Vector Field Histogram [1]. This method does not project a probability profile onto the grid. Only one cell is updated for each sensor reading. Rapid sampling provides a likelihood distribution over the grid. High certainty values should be obtained in cells near obstacles. This method is not used to create a map, but rather to provide robust low level obstacle avoidance.

Elfes more recent work regarding certainty grids involves a concept called active sensing [5]. This approach seeks to dynamically determine where it is worthwhile to direct the sensing apparatus, rather than rely on a fixed sensing strategy. This idea is obviously worthwhile, but many of the considerations are beyond the scope of this thesis. Some of the other proposed improvements to the certainty grid are very interesting. One idea is to actively keep extra information beyond the certainty that the cell is occupied. Examples include observability, reachability, color, reflectance, traversability, and connectedness. The generalized Occupancy Grid is called an Inference Grid. The extra fields of information are used to merge the *planner* and *mapper* into a single *planner/sensor*.

1.1.2 Obstacle lists

The success of the certainty grid methodology was largely based on the failures of alternative methods. It is difficult to create a representation to keep track of the edges of obstacles and account for uncertain sensor readings and possible conflicts. This difficult problem has been approached head on and solved in a number of cases [2][6]. A representative case is that of Larsson, Forsberg and Wenersson. The primary problem addressed was that of localization of the robot within the map. This can be done with certainty grids, but is not a procedure that can be performed in a practical amount of time during operation. Larsson and company instead keep track of obstacles and use the identified obstacles as landmarks in an attempt to bound the error in the

estimated location of the robot. This method provides the possibility of relocalizing the robot in real time.

It is slightly misleading to compare the studies directly as each uses a different type of sensor for mapping. Larsson and company use a time-of-flight laser, while the primary sensor in the certainty grid model was a sonar. It will later be noted how each of these sensors lend themselves to a certain type of strategy.

The method used by Larsson and company to account for poor sensor readings and conflicts involves keeping an Extended Kalman filter to estimate the coefficients representing each obstacle. In general, any type of non-linear estimation filter could be used. The Extended Kalman filter seems to particularly popular, perhaps due to the optimality of the Kalman filter on linear control problems. The state kept for each obstacle is a heading relative to the vehicle, and a distance. All obstacles are assumed to be flat walls. Obstacles which do not conform to the assumptions inherent in a flat wall are typically not recorded well, however given the indoor environment there are plenty of walls to provide landmarks. As the robot moves, each Kalman filter updates the relative heading and distance to each object. This provides a great deal of redundancy reducing the error in estimated heading and position of the robot.

It should be noted however, that this method creates very good maps of indoor walls, but does not do as well on obstacles that are not explicitly figured in the Kalman filter. It also seems to take a large amount of computational time, as a Kalman filter must be used for each feature and the results of the filters must be correlated. The certainty grid seems to provide a more conservative method of accounting for possible obstacles as it needs fewer assumptions. However, the power of a feature based map to possibly bound the error in heading and position, argues for its inclusion in a complete robotic system.

1.1.3 Local Mapping

In this thesis, local mapping refers to the real-time creation of a high resolution map of the immediate surroundings of the robot. This type of map is useful for complex maneuvers in highly cluttered environments. The intended purpose of a local map is

not to provide quick response to unknown obstacles. Emergency reactions to obstacles that suddenly enter the path of the robot should not wait on the creation of a map. Rather, the local map is useful after the emergency reaction has put the robot in a difficult position. A complex local maneuver such as a three point turn may require the robot to come very close to obstacles, especially if the purpose is to extricate the robot from a safety radius violation. The local map provides the information needed to do the complex planning required to extricate the robot from difficult situations.

Having a local map for emergency situations allows one to use it to improve the performance of the standard system. Typically a safety radius around the robot is used to keep the robot from coming to close to obstacles. If a local map is used it is possible to operate much closer to obstacles. This is very useful in indoor environments where a simple safety radius approach might keep the robot from traveling though doors.

Effective local mapping requires that the location of obstacles with respect to the robot be as accurate as possible. Over time an autonomous robot will accumulate errors in its absolute position, without the use of positioning aids such as beacons. It is therefore common to create the local map with respect to the position of the robot rather than an absolute coordinate system. The planning system must be able to query if movements are possible for the robot without hitting an obstacle. This query must be fast. Local maps do have the advantage that they have access to the most recent readings and knowledge that the readings are presently accurate with respect to the robot.

1.1.4 Global Mapping

A global map does not need to have the level of precision present in a local map. In practice it would be very difficult to create such a map with an autonomous robot. The main purpose of a global map is to allow effective long range planning. It is more important to allow a planning system to accurately determine if the robot can reach a distant place than to provide high resolution details of the obstacles along the path. A global map represents the permanent features of an area. Over time the map should converge to an accurate representation of the permanent features and ignore

transitory obstacles.

The area covered by a global map is much larger than that covered by a local map. It will therefore tend to have less resolution, use more memory, and possibly use secondary storage. A global map may also contain data that is out of date. It is most important that the large features are correct. Small details are not important as long as they do not affect possible planning operations. For example, the width of a doorway is not important as long it correctly represents that the robot can move through it.

1.2 Mission Objectives

Creating an autonomous robot does not leverage the main strength of computer systems. Adding a computer to sensors and actuators allows for repeated and precise movement. If one is able to provide a precise, constrained environment the programming of simple repeated actions makes good use of the strengths of computer systems. Programming a computer system to handle unconstrained environments where precision is of little help is more difficult. This is the task in front of one who wishes to create an autonomous robot.

1.2.1 Human Interaction

There are a number of non-autonomous robotic systems in use. These typically require a dedicated operator who controls the actuation of the robot while viewing the remote location through a video camera. Robots are used in these situations to keep humans from entering dangerous areas.

Human interaction is an important consideration in any robot system. Despite the term autonomous, the goal of a robotic system is to provide some service for an operator. An autonomous robot simply requires less conscience attention from the operator. The autonomous robotic system gives the operator a higher level interface than direct control of the actuators.

There are a large number of issues concerned with what type of control an operator

should be given and how to present it in an understandable manner. The benefits of an effective higher level interface are the possibility of a single operator controlling more robots, or spending more time on activities other than controlling the robot. These issues are important for useful robotic systems, but are beyond the scope of this thesis. The reliable operation of the systems considered in this thesis are considered necessary to implement such higher level interfaces.

1.2.2 Autonomy

The type of autonomy that is considered in this thesis is the ability to have the robot move to given locations without the intervention of an operator. This involves providing the robot with a number of waypoints which must be reached in order. Higher level planning systems have been created that allow more complex behavior given the basic ability of the robot to reliably reach waypoints.

The problem of reliably reaching waypoints is quite difficult. Simply creating a control system to allow the robot to follow a curve is a difficult task. To reach a waypoint a number of basic systems must work with high reliability. These include the method of estimating the current location of the robot, the control system for maintaining the robot on the desired path, the sensing and mapping system, and the planning system. Each of these systems depends on the ones below it as well as on all of the hardware. Given that Companion as well as most other autonomous robots are prototypes this is a long chain of dependencies upon which to rely.

1.3 Planning

To reach waypoints, a planning system must decide what path should be followed. Each planning algorithm attempts to find an optimal or near optimal path based on some criteria. A typical optimality criteria would be the length of the path, but a large number of alternatives are possible. A longer path may be faster due to less turning, or an alternate path may be considered better due to safety considerations.

1.3.1 Simple 2D

Simple two dimensional planning involves determining if there is a path between the current location and the goal waypoint. This type of planning finds a series of straight lines along which the robot can traverse without hitting any objects. This is done for long range planning on a global map. It can have a number of criteria for determining an optimal solution depending on what information is provided by the map. The typical method used to solve this problem involves the use of a hueristic search technique called A*. On Companion the planner uses a version of A* search to solve the problem. The hueristic used is the simple Euclidean distance to the goal. For further details on A* and the implementation on Companion refer to Terence Chow's Thesis [3].

Simple 2d planning is useful in environments where there is plenty of room to turn to the correct heading. This is the case in certain outdoor environments. If the robot can change heading without needing to translate in the (x, y) plane then this type of planning is sufficient.

1.3.2 Incorporating Heading

The straight line plan produced by a simple two dimensional planner does not take into account the need for the robot to turn to the correct heading. Some robots can turn to a heading by spinning without needing to move in the (x,y) plane. These robots present a much easier planning problem. For mobility systems which do not have this ability it is necessary to take into account the finite turning radius of the vehicle.

This type of planning needs an accurate local map as it involves complex maneuvers close to obstacles. In the absence of close obstacles, the vehicle can turn at its minimum turning radius to approach the heading and line between the current position and the goal. This is a simple method which works well in uncluttered outdoor environments. In an indoor environment, however, it may be necessary to move to a more open area before a complex maneuver allows the robot to change heading.

This type of planning is termed "3d planning" indicating that heading is considered as well as the (x, y) plane. This type of planning is highly dependent of the shape and mobility of the vehicle. It is however possible to generalize these attributes so that a single planner can be ported to multiple platforms. This problem is also solved by a variant of the A* search algorithm. The heuristic used in this case is much more complex. Transitions in the graph built to solve this problem involve simple vehicle movements rather than arbitrary straight line segments. This creates a much more complex solution space. The size of problems a 3d planner can solve in a practical amount of time is much smaller than that of a 2d planner.

Chapter 2

The Companion Robot Platform

2.1 Hardware Platform

The Companion robot uses the base of a motorized wheelchair for its mobility platform. Two deep cycle lead acid batteries power the rear wheels of the platform. Ackerman steering is used to reduce slip on the front wheels. The entire system is controlled with a joystick. The system used on Companion emulates the joystick to control the mobility platform. This allows the operator to use a joystick when the robotic system is not operating.

The robot has a number of features that make it well suited to effective and inexpensive experimentation in robotics. It features an onboard laptop computer networked via ethernet to an embedded processor. The real-time operating system QNX runs on both nodes, allowing for simple development of multi-process software systems using message passing as the standard interprocess communication method. The laptop also provides for ease of system upgrade. Replacing the original 486SX laptop with a newer 486DX4 with more memory significantly improved the performance of the robot and extended the useful life of the system. It should be noted that this upgrade was significantly easier and less expensive than the upgrade of an embedded processor, as would be required if an off the shelf laptop was not used. The laptop is also more versatile than a ground station as the possible communication bandwidth is much larger than provided by radio link.

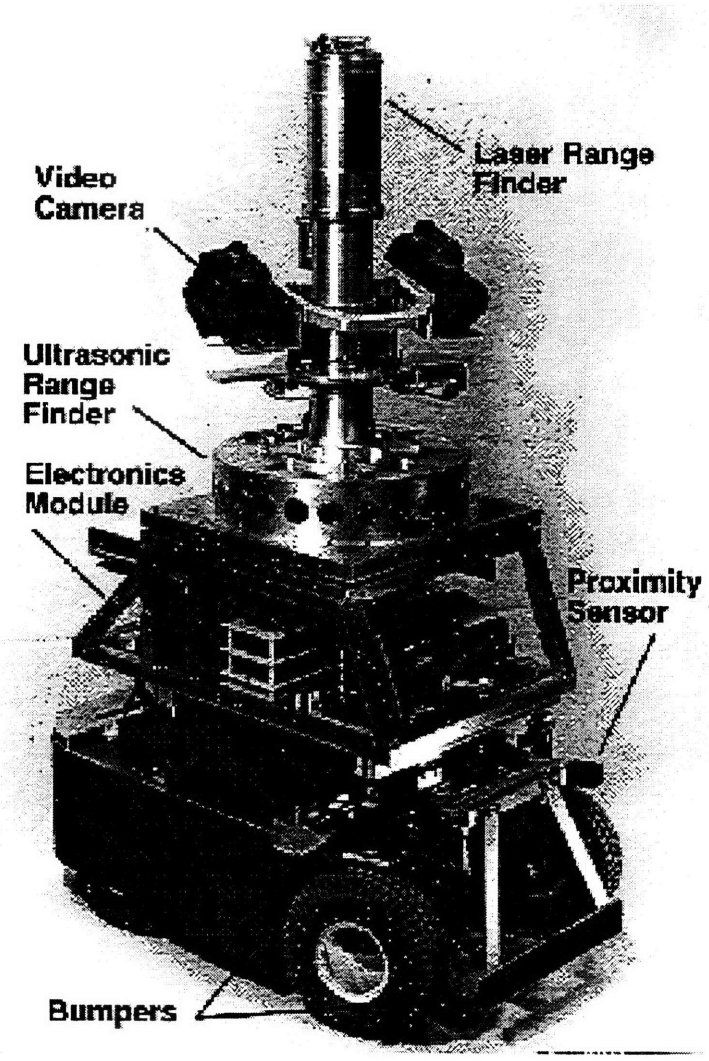


Figure 2-1: Companion

The Companion robot is a complex system. It will not be possible to detail the entire system but relevant information on the sensors, and computational systems will be presented in detail. The sensors can be split into two categories, trajectory sensors and obstacle sensors. Active trajectory sensors include an integrated rate gyro for yaw, two wheel encoders on the front wheels, and two steering pots on the front wheels. Active obstacle sensors include eight bumpers, eight infrared proximity detectors, 24 acoustic sensors, and a scanning laser range finder. Sensors that will not be used, but were part of the original design include: a compass, GPS, two inclonometers, and a vision system. These other sensors are either not completed, not integrated, or not debugged.

2.2 Computer System

The computer system present on Companion, shown in Figure 2-2, consists of a number of independent processors. A Winbook 486DX4 laptop computer provides a keyboard and screen on the vehicle as well as the main processor. Two embedded processors manage devices and lower level programs. One is a PC104 form factor 486SX with a math coprocessor. The laptop and embedded 486 are networked via ethernet and the QNX operating system. The final processor is a Little Giant. This processor is dedicated to the gyroscope and the steering pots. It communicates to the rest of the system via a standard RS232 serial line.

The other boards on the PC104 stack are an ethernet card, a DM406 digital I/O card, and a MEI motion control board. The DM406 I/O card interfaces to a custom built interrupt board. The interrupt board keeps track of the current state of the bumpers, infrared proximity detectors, and the two wheel encoders. When a change in state occurs, it sends an interrupt signaling that the new state should be read. The DM406 also has a number of D2A's which are used to control the steering and drive motors. The movement system is controlled by emulating the output of the joystick which came with the wheelchair.

The Little Giant is a stand alone board with digital and analog I/O, as well as

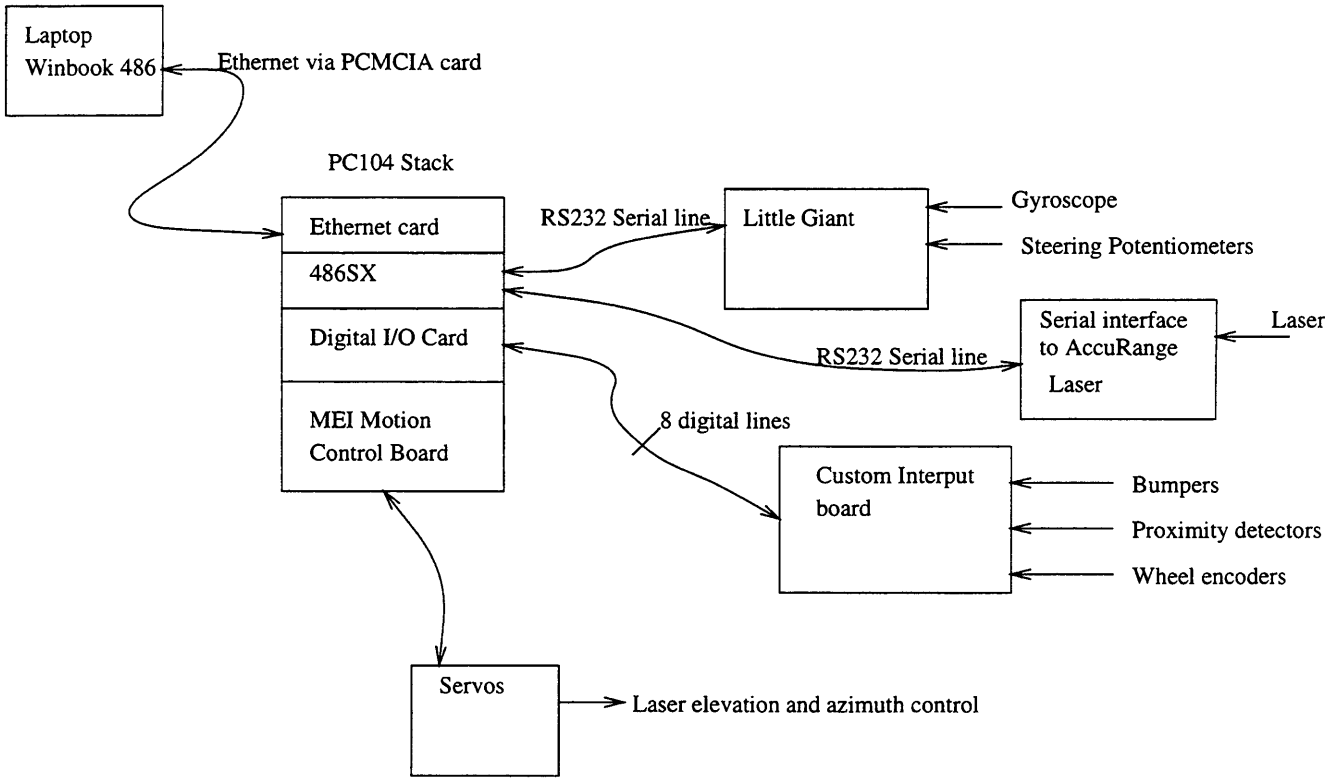


Figure 2-2: Computer System Hardware on Companion

built in serial ports. The A2D's are used to digitize the gyroscope and steering potentiometer output. The gyroscope is a rate based gyro, so the Little Giant is also used to integrate the rate to maintain the angular position. Drivers running on the PC104 486 can query the current angular position over the serial line. The steering potentiometers are also part of this system. Each response to a query over the serial line includes both the angular position and the current readings of the steering potentiometers.

2.3 Software System Architecture

The software system designed for Companion is shown in Figure 2-3. Terence Chow provides a detailed desription of all of the components of the software system in his thesis [3]. This section will provide a brief overview of the system architecture and implementation. The system is designed to run on a network of QNX nodes. QNX provides reliable message passing across the network as the main form of interprocess communication. In all cases except for certain communication between the *mapper* and the *search* processes the message passing provided by QNX was used, This allows those processes, to be distributed across the network in an arbitrary manner. Should processor time become a problem, adding additional processors to the system would allow for an immediate speed improvement by simply redistributing the processes.

The communication that does not use message passing involves access to the map created by the *mapper* and used by the *search* processes. In this case, shared memory is used, as message passing would exact delays in the inner loops of both the *search* processes and the *mapper*. This design choice forces all processes that use the shared memory to be on the same processor. The *planner*, *2d search*, *3d search*, and *mapper* processes are on the Winbook. The *trajectory*, *cycle*, *laser*, *sonar*, and *sound* processes are located on the 486 on the PC104 stack.

The software system uses two types of processes. One is a blocked process, which typically waits for a command from another process. When a command is received, the blocked process services the command and returns the answer. This is not a remote procedure call as the process giving the command does not need to wait for the reply. The other type of process is called a non-blocked process. This type of process either needs to be able to initiate or receive a command at any time, or must provide periodic services. It therefore cannot block while waiting for another process to reply. The user of the system can be modeled as a non-blocked process.

The blocked processes include the *2d search*, *3d search*, *laser*, and *sonar* processes. The search processes implement the search strategies described in chapter 1. The *sonar* and *laser* processes receive commands from the *mapper*. Each handles the

running of device drivers and packaging each reading with enough information to provide an update to the map. The extra information includes the state of the robot at the time of the reading, such as (x, y) position, heading, and any state specific to the sensor.

The *planner*, designed and implemented by Terrence Chow, receives commands from the user interface. It controls the two search processes. Waypoints are provided to the *2d search* and the subwaypoints found by the *2d search* are used as goals for the *3d search*. The two search routines use the map created by this thesis to plan the paths. It will been seen later how these two processes are able to use the same map representation for different problems and avoid the conflicts created by using the map during background modification by the *mapper*. The single commands returned to the *Planner* by the *3d search* are passed to the *trajectory* process.

The *trajectory* process implements a high level control loop to allow the robot to follow simple paths, such as straight lines and arcs. If more precise control over the mobility platform were possible this could be accomplished at a lower level. The *Trajectory* process could be improved with a more sophisticated controller. However, for the purposes of this thesis and the work currently being done on the robot the current controller is sufficient.

The final process is the *cycle* process. This is the most important process on the vehicle. It maintains all of the state for odometry and does dead reckoning. It checks for emergency conditions, such as proximity detector activation and bumper hits, and finally it controls the mobility platform. All of these tasks are done in a continuous tight loop. The commands that this process services include requests for the current state estimate of the robot, and commands to change the speed and curvature. The *cycle* process acting alone provides all of the features needed for a sophisticated remote control system. The rest of the system attempts to replace the human with a rudimentry control system for accomplishing the deceptively simple tasks of identifying obstacles and navigating around them to goal positions.

2.4 Environment Considerations

The operating environment under consideration for this control system is a crowded indoor environment. This type of environment provides many challenges that are not featured in an outdoor environment. Hallways present difficulties for acoustic sensors, and the close proximity of obstacles can create very difficult constraints on movement. These problems are addressed in this thesis. The indoor environment, however, also provides the possibility for some simplifications. It is assumed that the robot does not experience significant pitch or roll. This simplification reduces the amount of memory which must be used for a map and eases the operation of planning. The mobility platform of the Companion robot, is unable to tolerate a significant amount of pitch or roll, so this simplification does not greatly limit the practical uses of the system on this robot. Anomalies in this assumption, steep ramps, bumps, etc. will be modeled as obstacles where possible.

The main problem is dealing with the large range of possible objects and scenarios where the sensors can be fooled. Obstacles need not be convex, solid, acoustically reflective, etc. The fusion of sensor information to create the obstacle map must deal with conflicting information from multiple sensors.

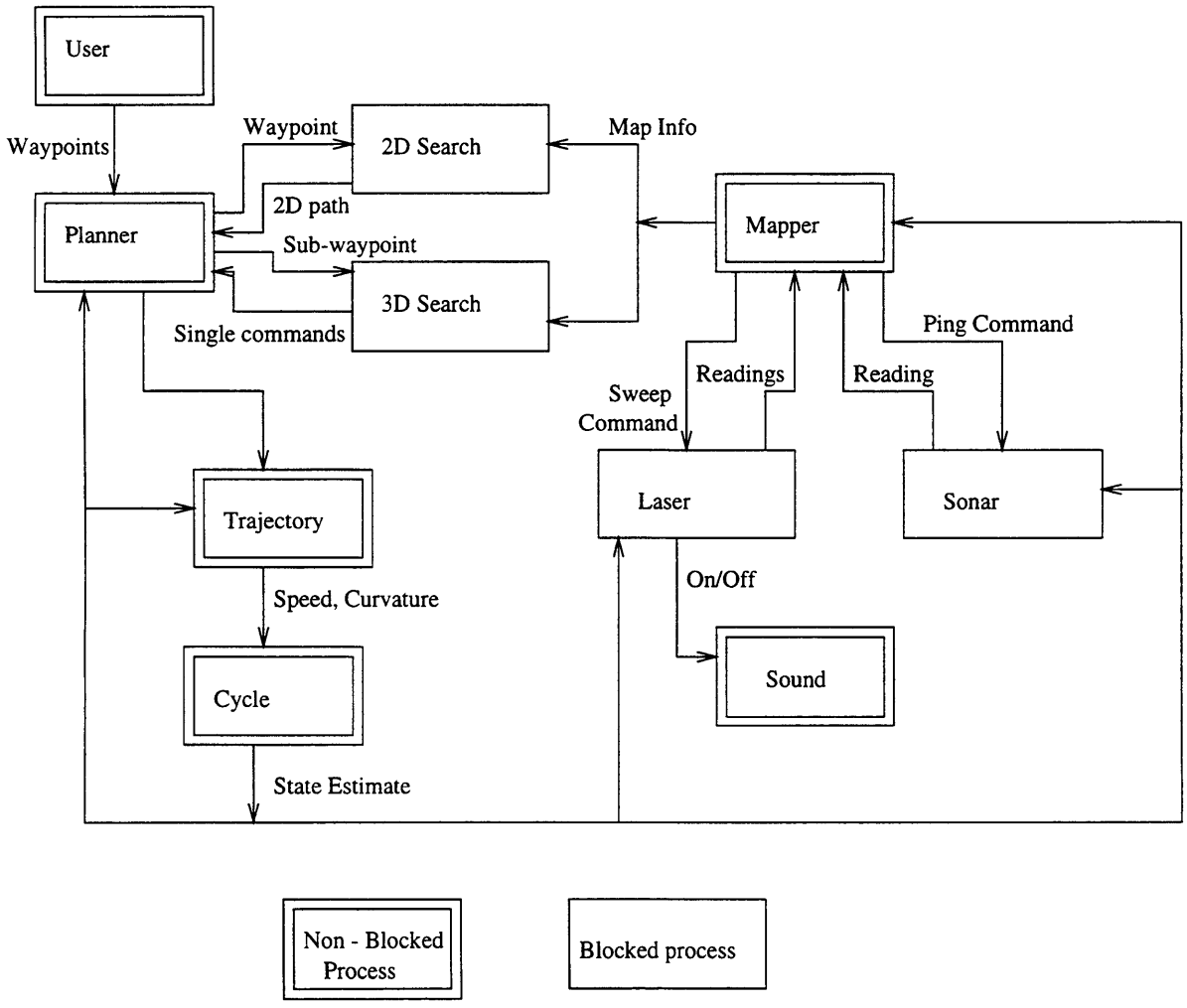


Figure 2-3: Software System Hardware on Companion

Chapter 3

Individual Sensors

It is useful to have an appropriate model of each sensor. This chapter will explore the information content present in a single reading of each sensor. Issues relating to how these models should be implemented in practice will be presented in another chapter. There is no claim that the models presented are the only possible formulation. Effort has been expended to make the models complete and clear rather than elegant. For each sensor, the model should address four questions.

1. What is the useful information this sensor provides?
2. What errors are possible in that information?
3. Under what circumstances can the sensor be used?
4. What information does the sensor *not* provide?

The last question may seem redundant, or overly broad. From an engineering standpoint, however, it is possibly the most important. The purpose of using multiple sensors for fusion is to leverage the best information from each sensor rather than use complex methods to derive information only marginally provided by the sensor.

3.1 The Sensors

Companion has a large number of sensors. For the purpose of map creation the primary sensors are the ring of 24 ultrasonic transducers and the scanning laser range

finder. The gyroscope, wheel encoders and steering potentiometers together provide the information for estimating the position of the robot with dead reckoning. The bumpers and infrared proximity detectors are primarily used for emergency reaction to avoid collisions or stop the robot after a collision has occurred. These sensors can be incorporated into the map, however they only provide information after a failure has occurred and are not considered a primary source of information.

3.2 Scanning Laser Range Finder

The scanning laser range finder system was effectively created in the lab. The laser range finder, a commercial product sold by Acuity Research Inc, is an AccuRange 2000. The scanning capability was added by a two axis deflection mirror. The deflection mirror can be rotated a full 360 degrees of azimuth and about 75 degrees of elevation. This gives a coverage area which is the complement of a cone with apex at the mirror as shown in Figure 3-1.

The interaction of the laser and the mechanical system for positioning the mirror make this a complex sensor. The operating principle of the laser will be explained and relevant specifications will be provided. The mechanical assembly and control system for the mirror will then be presented. Finally the software interface to the system will be presented.

3.2.1 Physical Properties

The laser determines range by collecting the energy reflected back from the beam. When the energy collected passes a threshold the laser is deactivated. Because of the laser's deactivation the energy at the collection point will become lower causing the laser to be re-energized. This negative feedback loop creates a square wave with a frequency proportional to the reflection distance. This frequency is converted by the serial interface to a range along with information about the laser's temperature and the ambient light which may affect the reading.

The serial line interface to the laser allows one to select an update rate vs. ranging

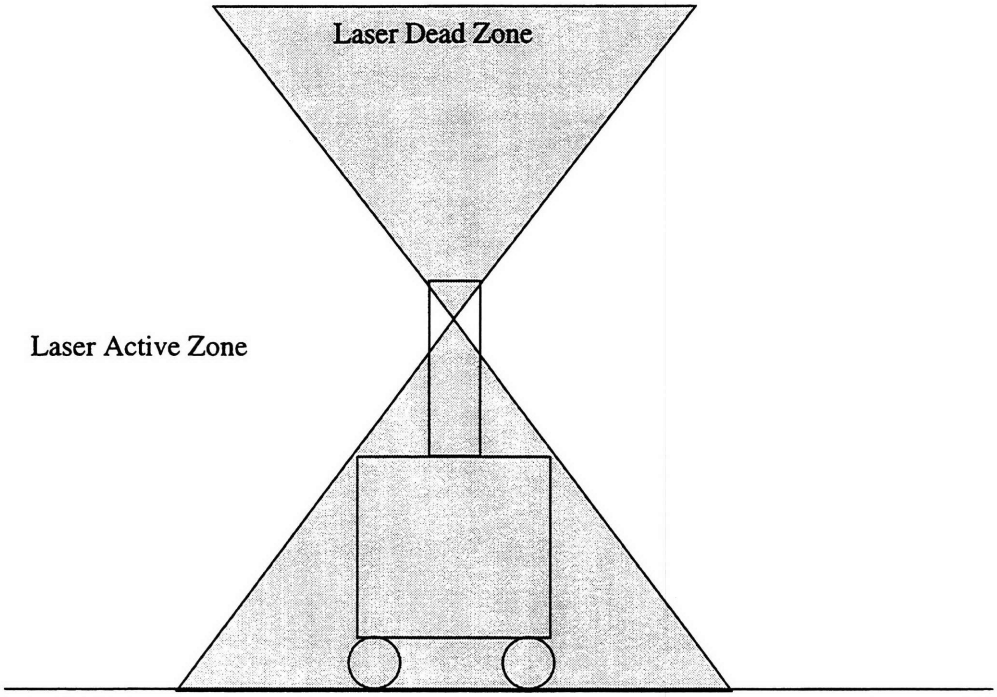


Figure 3-1: Laser Coverage

tradeoff. While the laser is active the serial interface provides range measurements at its maximum pace. The device driver created to read from the serial interface attempts to correlate the ranges provided to equal spacings along the sweep of the laser. Each range return is paired with a reading of the mirror position. Error can occur between the mirror position assigned to a reading and the actual position of the mirror at the time of the reading. It is possible to remove this error by only taking readings when the mirror is stopped, however this significantly reduces the throughput of the laser readings.

The mirror positioning uses two precision motors with built-in encoders and 3 limit switches. The motors use servos controlled by an MEI motion control board. The motion control board allows for a trapaziodal velocity profile, and accurate positioning measurement relative to the limit switches.

3.2.2 Software Interface

The software interface provided to higher level software allows for sweeps in azimuth and elevation. The number of laser readings taken during the sweep can be set from 0 to 100. The driver attempts to maintain an equal radial spacing between the readings. The return value of a single laser reading provides the distance measured, the azimuthal position and the elevation of the mirror, and the estimated (x, y) position and heading of the robot at the time of the reading.

This interface is provided by a blocking process. This process manages the sweep of the laser. It also correlates the range measurements to azimuth, and elevation positions, as well as querying the *cycle* process for the current position estimate of the robot. Finally it also provides a safety feature by turning on the sound process whenever the laser is active.

3.2.3 Characterization of the Errors

It should be made clear that this laser does not detect the nearest object which is an obstacle to the robot at a certain azimuthal angle. It can overshoot low obstacles near the vehicle. It can miss poles. It gives little information about whether an area or volume is clear. It does, however, provide accurate measurements of the boundaries of those obstacles it hits. Figure 3-2 illustrates the obstacles that the laser can miss during normal operation. It also illustrates a number of cases where misinterpeting the laser reading as the edge of the nearest obstacle would cause the robot to exhibit dangerous behavior.

The information provided by the laser gives relatively good angular accuracy (compared to the sonar), and very good range accuracy. This makes it useful for finding the borders of obstacles and openings such as doors. The laser is useful in hallways as it does not experience the specular reflection that is found in sonar. It can be reflected by mirrors, however that type of error can sometimes be accomodated with a prefilter called the Hough transform which will be introduced later.

3.3 Sonar

3.3.1 Physical Properties

The Polaroid acoustic sensor provides a time of flight measurement to the nearest obstacle within a certain volume. A transducer creates an ultrasonic chirp, here called a "ping", which travels outward from the sensor. After the transducer has been allowed to settle it can be used to sense a reflection of the ping. A counter keeps track of the time between the ping and return. The settle time of the transducer determines the minimum distance that can be sensed. The maximum distance is determined by the timing of the counter circuit and the limitations of the Polaroid detection circuitry (about 35 feet).

3.3.2 Software Interface

The sonar interface is provided by a blocked process. it waits for a command to ping a specific sonar on the ring. After pinging that sonar and receiving the response the sonar range is returned along with the estimated state of the robot at the time of the reading. The sonar ring is only able to ping a single sonar at a time. This is forced by the fact that there is only one counter circuit to track the time between the sending and return of a ping. If additional counter circuits were created it is possible more sonars could ping at one time. However, that would add the complication of dealing with reflections from the other sonars.

3.3.3 Characterization of the Errors

Specular reflection can cause the sonar to error in the amount of free volume it indicates. This is the case where the chirp hits an acoustically reflective surface and is reflected away rather than back toward the sensor as shown in Figure 3-3.

It should be clear that the sonar cannot provide definitive measurement of free area as it is subject to specular reflection. It also gives little information about where objects are. The sonar is best at providing *confirmation* that a volume is obstacle

free. Providing a return occurs and is not the result of a reflection then a cone of that height directed out from the sonar is free of obstacles.

3.4 Bumpers

The bumpers allow one to determine if the vehicle is in contact with an obstacle. If a bumper is depressed the location of an obstacle is known to be touching the robot at that bumper. The error in the absolute position of the robot and the uncertainty of where along the bumper comprise the positional errors possible.

The bumpers are mainly used only as reactive sensors. If a bumper is hit the robot stops. Under expected operating conditions a bumper should not be hit. Instead the robot should react due to higher level sensors and avoid contact with obstacles.

This sensor provides good information relative to the robot. However, under most practical circumstances if a bumper has been hit then there is a good chance that the error in position is great. Therefore the bumpers give little information about the absolute position of any object. The bumpers can be incorporated into the mapping strategy developed in this thesis, however the extra capability provide is considered marginal compared to improvements in the main sensors.

3.5 Infrared Proximity Detectors

The infrared Proximity detectors provide a binary valued return that indicates if an object is within a set straight line distance of the sensor. Despite the manufacturer's claims, proximity detectors are sensitive to the type of material which reflects the infrared beam. A highly reflective material will set off the sensor at a greater distance than a dull material.

The proximity detectors are used primarily as a reactive sensor much as the bumpers. However, the proximity detectors provide enough information to be useful in the location of obstacles and in moving. It is quite possible that a navigation plan would call for following a wall just outside the distance that sets off a proximity detector.

The detector could be used in a fast control loop for wall following. It is also possible that the navigation system could use the proximity detectors for determining the edges of obstacles by maneuvering close to the obstacles.

Two of the proximity detectors are mounted such that they turn with the front wheels. This allows for reactive commands when unexpected objects are in the path of the wheels. However, it greatly increases the uncertainty of the sensor information. The steering potentiometers must be used to determine where the wheels face, adding another source of error to the position of obstacles sensed by the front two proximity detectors. Companion uses these sensors only in a reactive capacity. The other possible uses are not exploited as the laser and sonar already provide those capabilities in a simpler form.

3.6 Wheel Encoders

The front two wheels have incremental optical encoders which determine the distance each has traveled. The encoders with their current counting circuit yeild 580 counts per meter of forward movement. Ackerman steering is used on the robot, meaning that unless the robot is moving forward the two wheels rotate at different rates. Given this information the two wheel encoders are sufficient to provide odometry for the robot.

There are a number of possible errors in information provided by the wheel encoders and its use for odometry. Tire pressure will change the distance traveled per rotation. Slip on the wheel can cause the encoder to indicate movement that does not correspond to translation of the robot. This type of slip happens when the steering angle is changed while the robot is stationary, however the stationary case is handled in the software to improve the odometry. It is also possible for the vehicle to slide, dragging the wheel without rotation. This type of movement will fail to be accounted for in odometry that relies only on the wheel encoders. The odometry implemented for Companion makes use of the gyro and steering potentiometers.

3.7 Integrated Rate Gyroscope

The angular position of Companion is determined by integrating the rate provided by the "gyro", an inertial angular rate sensor. The integration takes place on the Little Giant processor. The 12 bit A2D on the Little Giant is used to digitize the output of the gyro. Software written for the system integrates the angular position from the digital samples. A calibration routine is used to determine the base sample that indicates no angular motion. The operator must guarantee that the robot does not move while the calibration routine is running.

The angular position of the gyro system (with integration) drifts over time. This drift has been measured to be about 4 degrees per hour. For long-term operation it is critical to have a compass or some other means of finding absolute heading to bound the accumulated error in the gyro.

3.8 Steering Potentiometers

Each of the front wheels on Companion has a potentiometer to determine the angle of that wheel. The wheel angles are not independent, so the two readings provide redundant information on the current curvature of the robot's path. It should be noted that commanding the steering to a specific angle is more error prone than the reading of that angle. The system is easily observable but difficult to control. The joystick controller provided with the wheelchair has hysteresis. This is a beneficial feature when the control is being provided by a human, but becomes a great difficulty in attempting to provide steering control with an algorithm. A closed loop control system was not implemented because a threshold chamge in commanded steering angle must be exceeded in order for steering to actually occur. The *trajectory* process provides a higher level of closed loop control with a path following algorithm which gives satisfactory performance in the presence of poor control over the steering angle.

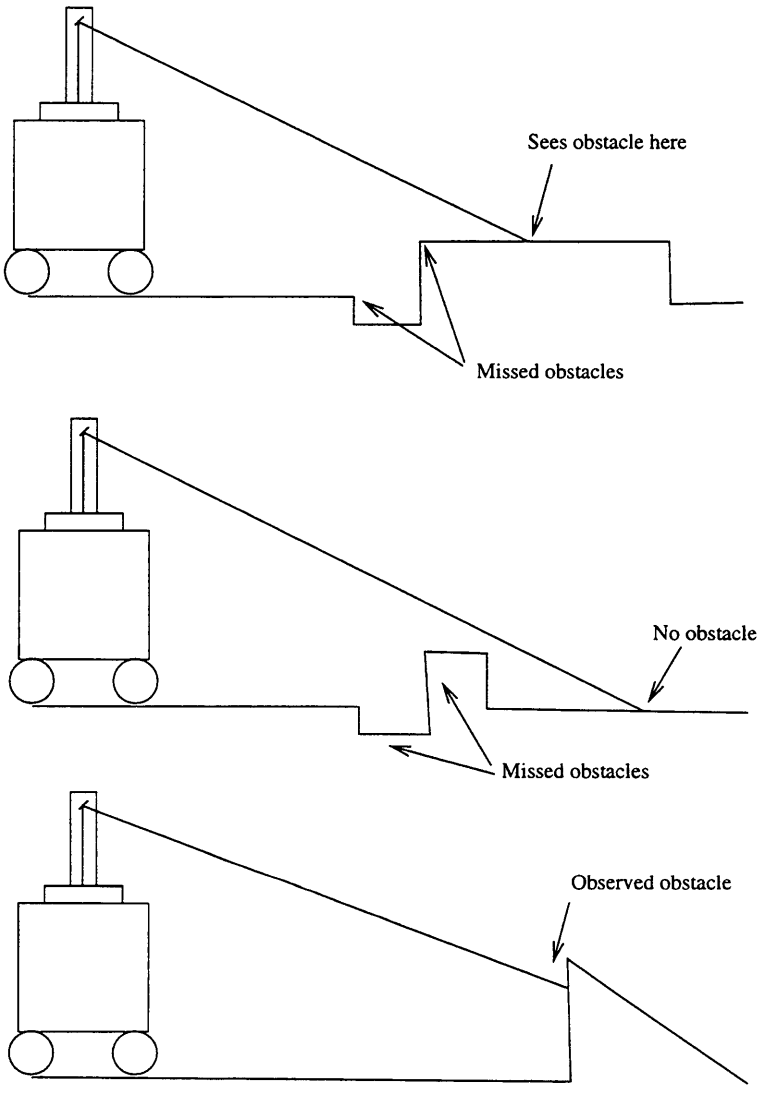


Figure 3-2: Laser Readings with Possible Errors

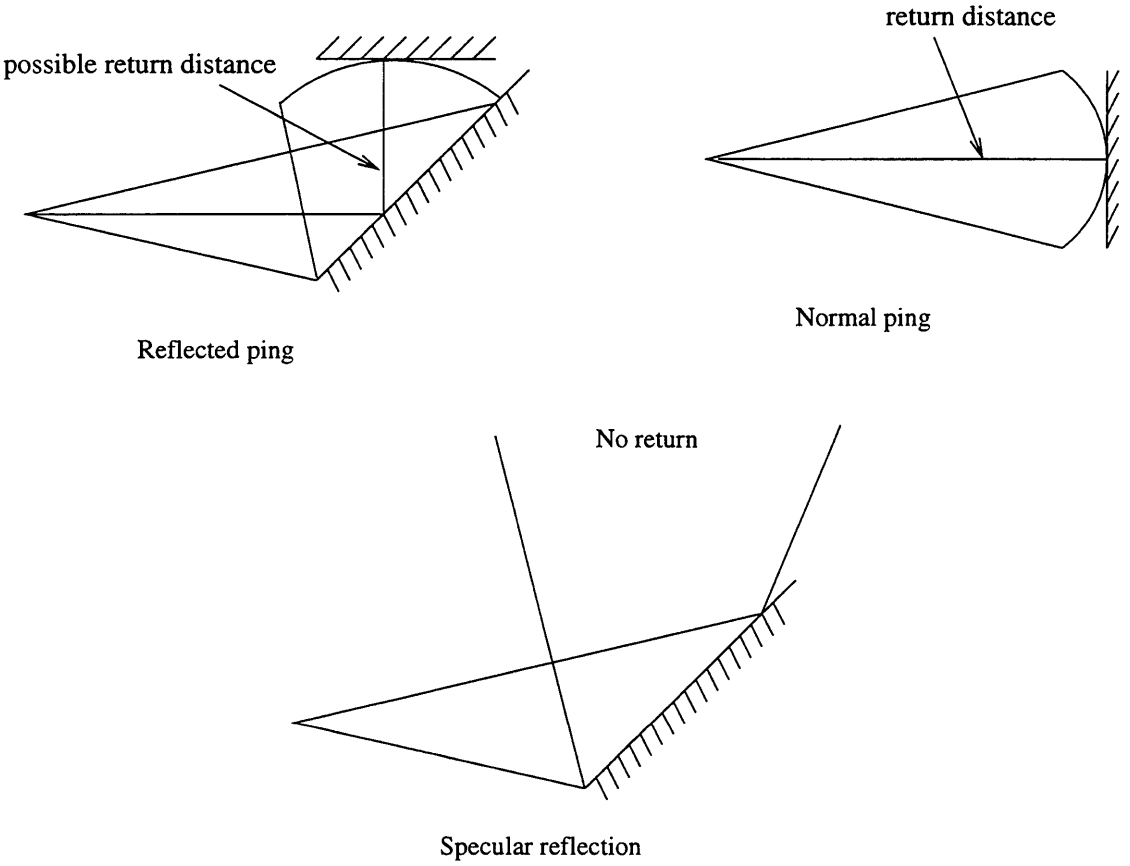


Figure 3-3: Sonar reflection

Chapter 4

Sensor Fusion

The practice of fusing sensor readings to create a robust model of the local area provides a number of advantages. The fusion routine can be used to tolerate errors in the readings. Less expensive sensors can be used on the robot platform, and leveraging the strengths of each sensor can reduce the latency in incorporating the sensor readings into the map. It is often the case that a single sensor does not provide enough information to give a robust model of the surroundings. Even expensive sensors will not provide robust error free indication of the obstacles.

4.1 Error Characterization

4.1.1 The Robot

The robot itself provides information through dead reckoning. By definition, if the robot is in a certain location then that location is obstacle free. The collection of these measurements will generate relative errors at the same rate that dead reckoning errors accumulate. These errors accumulate over time from the original reading.

If one is creating a map in absolute coordinates then the error in new readings grows proportionally to the dead reckoning errors. If instead the map is created relative to the robot, the errors in the readings are a function of the distance traveled since they were taken. This allows one to use such readings until the distance traveled from them

exceeds an allowable error margin. It will be seen that this distinction is important for determining how the reading should be used in a local map versus a global map.

4.1.2 Sonar

The sonar can experience specular reflection. This occurs when the acoustic wave produced by the sonar hits a surface that reflects it away rather than back toward the sonar tranducer. Typically this causes the sonar to give maximum returns when pointed at objects such as angled walls. However it is also possible for the reflected wave to eventually hit an obstacle and reflect back through the same path providing a nonmaximum reading that overestimates the free space to the nearest obstacle as shown in Figure 3-3.

This problem is much less likely to happen if the range returned is small. Any nonmaximum return precludes specular reflection without a return, and a short range reduces the likelihood that multiple reflections happened along the path. It is possible to partially account for reflection by making a spatial probability distribution that the area is clear. Another method is to keep readings from different angles separate. This would allow one to post process a large number of readings to determine if the range was due to reflection.

4.1.3 Laser

The laser can err in the absolute angle as was noted earlier. It is also possible for the laser to meet a reflective surface causing unknown effects to the range information. The laser can also experience bad readings due to heat. Finally the serial interface can drop readings due to noise on the line. This can cause a number of unforeseen problems.

Currently the raw laser readings are not considered as reliable as the sonar. There are a number of cases where the readings from the laser do not correspond to obstacles. This happens often at the ends of the laser sweep. This is possibly due to the high acceleration at the ends of the motion. The motion control board provides a trapezoidal

velocity profile. The laser range is provided by creating a feedback loop with the range as a linear component. If the range changes suddenly during this loop it is possible that unexpected ranges are returned.

Another possibility is that a yet undiscovered system bug is corrupting the readings. The laser is a recent addition to the robot and has therefore not been tested under real conditions for the same length of time as the other sensors. Regardless of the cause, these errors must be taken into account when the laser is used in mapping.

4.2 Tolerating Errors

Once one has recognized that errors are inevitable in any system, effort should be made to tolerate the errors which cannot be prevented. In the case of sensor readings it is worthwhile to attempt to identify a bad sensor reading before it is used to modify the map.

4.2.1 Fail Fast

A standard method in creating a robust system is to have failures manifest quickly. One wishes to recognize that something has failed as early as possible so measures can be taken to correct the problem. The Hough Transform can be used to do this for the readings in a laser sweep. This method attempts to correlate individual readings to possible obstacles. In a single sweep, it degenerates to grouping readings together by the obstacles that are hit. Obviously this method makes some assumptions on the type of obstacles that will be seen. However, those assumptions can be modified by the designer and are not very restrictive.

After the Hough transform has created a number of possible obstacles and grouped the readings that lend the most support to those obstacles it is possible that there remain some readings that do not lend support to any obstacles. These readings are considered spurious and not used in mapping.The Hough transform uses knowledge about the expected obstacles to identify single reading failures.

There is one modification to the Hough transform that improves its performance

on data collected from the laser range finder. This modification takes into account the greater frequency of hits at closer ranges. When the laser is fired with equal radial spacing, closer ranges will be hit more frequently as shown in Figure 4-1. The Weighted Hough transform weights readings with larger ranges more to account for the skewed distribution of readings.

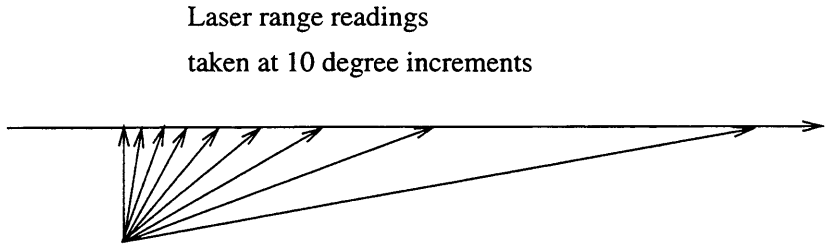


Figure 4-1: Distribution of ranges with equal radial spacing

Another possible error does not involve the failure of any of the sensors. Rather it is possible for a fundamental assumption to fail. One such possibility is quick moving objects which one does not wish to incorporate into the map. An example would be incorporating the developer walking near the robot as a permanent feature of the landscape.

If it is possible to recognize that readings are being received from obstacles which do not remain in place then it would be preferable not to include those in a global map. Those readings should still be included in a local map as one does not wish to run down people simply because they are not part of the landscape.

It is also possible to recognize that the absolute position estimate of the robot is in error and correct it through the use of previously identified landmarks or beacons set up for the purpose of localization. In these cases, it is useful to be able to reset the position of the robot and in some manner either update or discount the most recent readings from the other sensors.

4.2.2 Error Masking

Another method of tolerating errors is to make the system of incorporating the readings into the map robust versus bad readings. Typically this approach means that it will be necessary to take more readings to create a useful view of the local area.

The use of certainty grids provides for error masking by allowing conflicting readings to keep the cell in a state of uncertainty. This is good since if one reading indicates that a cell is open and another indicates that it has an obstacle it is not possible without other information to determine which is in error. Typically the probability profiles allow one to weight the more likely circumstance. For example a laser hit is not easily contradicted by a single sonar reading since the sonar might be experience specular reflection. An alternate approach is to use a Kalman filter to determine what weight should be given to the new reading versus the already created model.

Neither of these approaches ever determines which readings were in error. Instead they rely on the assumption that the majority of the readings will be good. The consistency among the correct readings creates a stable model of the space or the obstacles, while the incorrect readings add a small amount of noise in the form of either uncertain grid cells or slight changes in obstacle locations.

4.3 Contrasting Certainty Grids with Obstacle Feature Lists

4.3.1 Understandability

The understandability of the approach is important from an engineering point of view. It is not easy for one to build something that is not well understood. The robustness of the certainty grid approach is more apparent. It also does not depend on the type of obstacles that are being mapped. The obstacle list approach to mapping is understandable, but the methods of making it robust are not as simple. The method of adding a Kalman filter to arbitrate between the current model and the new readings

requires a robot specific design. It also requires a number of assumptions which are not needed in the certainty grid. Unanticipated obstacles can severely affect the effectiveness of using a filter on an obstacle list.

4.3.2 Effectiveness

Both the certainty grid and the obstacle list approach to mapping are effective. Each, however provides maximum benefit in a different area. The obstacle list provides an excellent method of accurately determining the location of recognizable objects such as walls and poles. It can also be used effectively to maintain the location of the robot with greater accuracy. The certainty grid approach is more versatile. It does not require as many assumptions about the types of obstacles as the obstacle list method. The certainty grid also allows one to implement either a conservative or aggressive approach to interpreting the sensors.

The certainty grid does give up one powerful advantage that can be used with an obstacle list. There is no real concept of an obstacle above the level of a cell in a certainty grid. With an obstacle list it is possible to identify obstacles and associate them with sensor readings. Identifying obstacles with sensor readings is the main job of an obstacle list approach. By identifying the cause of a particular sensor reading it is possible to use the obstacles to better track the position of the robot as it moves. just as the gyro should be recalibrated regularly, it is useful to have some method to determine the absolute position of the robot. The information in a certainty grid does not make the task of identifying particular obstacles simple.

The inability of the certainty grid to determine specific obstacles does affect its usefulness for specifying the position of the robot. However, it should be noted that robot position can be solved with other methods. Further the need of the obstacle list approach to find the edges of obstacles identifies part of its weakness. The sensors present on Companion are versatile, yet cannot guarantee the identification of the edges of obstacles. The laser may see the top of an obstacle, missing its edge. The sonar does better at identifying open space than in finding the edge of an obstacle. To make a usable map, the certainty grid need not make the assumption that the sensors

will see all of the obstacles.

The certainty grid is more robust than an obstacle list with a Kalman filter. The Kalman filter can only be robust against conflicts which are accounted for in the design of the filter. The certainty grid is inherently robust due to the distribution of the information and the stochastic interpretation of the data. A Kalman filter would need to recognize when a specular reflection has occured. The certainty grid can still be used even if specular reflections are incorporated into the map, though it works better if they are not. Each approach can deal with specular reflection, but the certainty grid does it by default. The certainty grid can deal with spurious errors that involve completely wrong readings rather than just noise added to a correct reading. It is possible that a Kalman filter design could do this, but it is not apparent that that is the case.

4.3.3 Extensibility

The certainty grid approach is more easily extensible than the obstacle list. The certainty grid allows for the independence of the different sensors. An obstacle list typically takes the dependencies of the sensors into account to derive a robust estimate of the location of each obstacle. A change in the system can cause the need for a large modification of the filter used in an obstacle system, while a certainty grid is likely to need very little change. The versatility of the certainty grid is also important for using the approach across multiple platforms.

Chapter 5

Mapper Architecture

The algorithm for creating the map has a number of requirements. These requirements come from the structure of the planning algorithm, the inherent limitations of memory size and computational speed, the quality of information provided by the sensors, and the intended use of the map. This chapter will explore these requirements. It will identify the reasons behind each requirement and help to prioritize which are most important for the Companion robot. It will also introduce the architecture which was created for the robot.

5.1 Requirements

The primary requirements vary depending on how the map will be used. In general one can sacrifice precision or accuracy for speed and memory savings. The algorithm can be designed as an online algorithm incrementally updating the map with each sensor reading or as a batch algorithm working on a large number of readings at once. Two design approaches will be used in examples to clarify how the requirements affect both the design and implementation.

The first approach is the use of a certainty grid. This method quantizes the space into a grid. Each grid cell contains a certainty value which expresses the level of certainty that the area is occupied. One choice for certainty values is to use probabilities. This allows one to use Bayes' rule to update the values in each cell given a spatial

probability function for each sensor. The method of determining if it is possible for the robot to be in a specific position involves looking at all the cells that the robot overlaps at that position. If any of the cells indicates an obstacle than that position is not legal. There are a large number of variations on the basic idea of a certainty grid. These will be explored throughout this chapter.

The other example is an obstacle list approach, typically using Kalman filters. This approach uses sensor readings to identify the edges of obstacles, and a filter to determine the weight to give sensor readings versus the current location of each obstacle. A list of polygons, or sometimes just line segments representing the edges of obstacles is created. A line crossing algorithm is then used to determine if the robot overlaps any of the obstacles. This approach also has a great number of variations, which will be explored throughout the chapter.

5.1.1 Partitioning Space

The main task of the *mapper* is to partition space into empty and obscured regions. The algorithm should allow one to label regions of space as either free or occupied. The only functional requirement the planner imposes on the mapper is to identify if the robot can be in a specific position. Obviously this requirement can be satisfied if an accurate high resolution map is obtained. However, high resolution is not necessary everywhere on the map. In the case of a grid the cell size is inversely proportional to the resolution of the information. This is the primary tradeoff in a certainty grid, as updates will tend to be done in constant time.

In an obstacle list the number of bits used to keep the position of the obstacles determines the resolution. Typically however the locations of obstacles have high precision. Memory use is not the main concern in implementing the obstacle approach but rather update latency. It is possible to tradeoff the accuracy of obstacle locations to reduce latency.

The partition depends on both the map and the shape and orientation of the robot. In the case of a certainty grid a single cell which indicates obstacle creates a polygon around that cell as shown in Figure 5-1. Similarly when line segments are used,

polygons along the edge of the segments are the areas that cannot be entered. An obstacle need not be solid or visually apparent to humans on the map. It simply must model the area well enough to accurately answer if the robot can occupy the space near it.

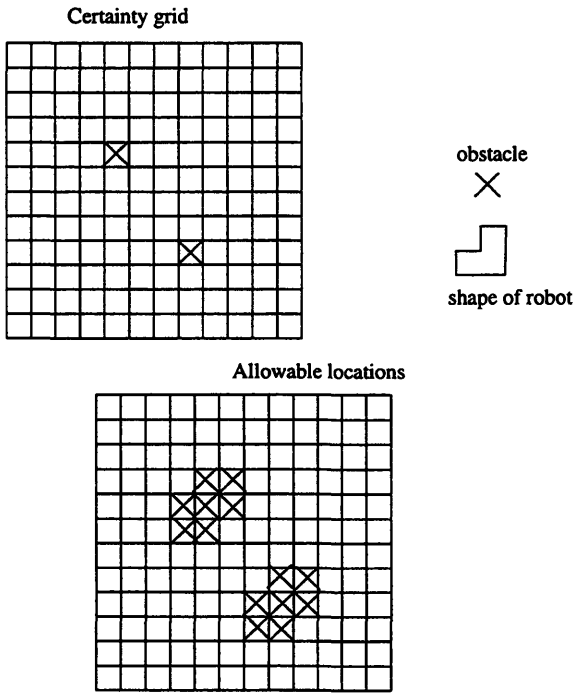


Figure 5-1: Obstacle Expansion

5.1.2 Local Map

A local map provides high resolution detail around the immediate area of the robot. This type of map is used by the 3d planner. It allows the planner the ability to determine whether the vehicle is in a legal area depending on the heading as well as the (x, y) position. This query should be fast as the local map is heavily used in planning operations that need to have low latency.

The *mapper* should also use the most recent sensor readings for a local map. Old readings may reflect objects that no longer exist or free area that has since become occupied. The latency between sensor readings and incorporating those readings

into the map must be small if the map is to be a timely representation of the local space. This suggests that the mapping algorithm should be formulated as an on-line algorithm.

A local map should be small enough to fit in main memory. The map should be roughly centered on the robot at all times. The operation of centering the map on the robot as the robot moves must be fast. This requirement is mainly of concern if one is using the certainty grid approach. If a grid is to be used one must either move the data to allow the robot to be roughly centered or use a level of indirection before indexing into the grid. In the case of an obstacle list, one wants to only consider the obstacles that are visible.

Finally, a local map is sensitive to the precise location of the robot. In this respect the obstacle list approach allows for a benefit that is not feasible with a certainty grid. With an obstacle list approach, it is possible to use the known obstacles to reduce the accumulated error in the postion estimated by odometry.

5.1.3 Global Map

A global map provides an accurate representation of the area traveled given a sensor history. It need not be an on-line algorithm, and need not give priority to recent readings. These maps tend to contain permanent features of the terrain. Moving obstacles should not appear in the map or should be added and removed only after many sensor readings.

These maps will take up more space, and possibly go to secondary storage. The resolution of the map need not be very high as long range planning does not take local maneuvers into account. The global map does not need to be centered on the robot. However, it must allow for the robot to travel in any direction. It is therefore difficult to place bounds on the area the map will cover.

The difficulty in implementing a global map with the obstacle list approach is that the state on which the filter must act grows unbounded. The size of this state is not the problem as the obstacle list encodes it very efficiently. The problem is that the time complexity of the algorithm is directly proportional to the size of the state. To

deal with this problem, one must define a subset of the state on which to act for incorporating a single sensor reading. To provide a robust update of the state on an obstacle list a filter is used to arbitrate updating the location of obstacles, the location of the robot, and the introduction of new obstacles. An example of such a filter, which will be explored later, is the extended Kalman filter. Using such a filter will tend to increase the latency of updates, but increase the accuracy and reliability.

The certainty grid must either have known bounds for the edges of the map or be implemented in such a way that the bounds can be increased. This approach also has a problem in that the space grows unbounded. The time complexity of updating the map remains constant, but the size of the state in a certainty grid is much larger than that of an obstacle list. In the case of a global map, the area that a cell represents can be set larger to save on memory.

The certainty grid does a very good job of implementing the type of map update required for a global map. A large number of prior sensor readings will keep cells from being changed quickly due to current sensor readings. This is good for long term global mapping, but detrimental to local mapping.

5.2 Recursive Formulation

A recursive map formulation takes advantage of every sensor reading. It makes the assumption that only stationary objects are to be mapped. It is very good for finding the regions which may be traversed and long term mapping. It also does not need to take into account the order of the data readings, and can be used as either an online algorithm or a batch algorithm. This type of algorithm is naturally implemented by a certainty grid and is appropriate for global mapping.

The operations that must be defined to specify a recursive formulation are adding a single sensor reading to a default map, and the merging of two overlapping maps. The certainty grid can implement this by using probabilities for the certainties and Bayes' rule to update each cell. This is the type of certainty grid created by Elfes and Moravec [4][7].

To implement this approach with an obstacle list it is typical to create a multistage algorithm. Sensor readings are first associated with an obstacle or create a new obstacle for the list. Then a filter, such as an extended Kalman filter updates the state of the obstacle and the robot. This involves determining a new estimate of the absolute location for the robot. The obstacles are estimated with respect to the robot. At the end the readings are merged to provide a single new estimate for the location of the robot. Thus each obstacle location is tracked with respect to the robot with an active control algorithm. Using an extended Kalman filter with this approach will implement a recursive formulation. This approach was explored by Larsson and company, and Borthwick and Durrant-Whyte [6][2].

5.3 Dynamic Stability

A dynamic stability approach sacrifices the convergence properties of the recursive formulation to allow for the timely addition of moving obstacles to the map. This type of approach relies more on the correct operation of the sensors. Poor sensor performance will cause this type of strategy to create false obstacles and remove existing obstacles by mistake. The tradeoff is a better picture of the obstacles at the present for a less robust picture. An extreme example of the dynamic stability approach is the Vector Field Histogram created by Borenstein [1].

Recursion may still be used in ta dynamic stability algorithm, however this approach weighs the most recent readings above those obtained earlier. It is a more appropriate method for local mapping. A certainty grid can implement this approach, but not by using Bayes' rule. An obstacle list can also implement this approach, by using an appropriate filter for the sensor readings.

5.4 Default Map States

5.4.1 Open or Closed Assumption

A map must start with a default state in the absence of other information. This default state can be uncertainty, but unless the planning algorithm takes uncertainty into account it will eventually be mapped to either occupied or unoccupied. The choice of default state determines how conservative the planner will be in determining how to move. A nongrid strategy typically uses a default state of empty and attempts to identify the borders of all obstacles.

A default state of occupied takes the approach that sensor readings must determine if the space around the robot is free. This approach is very conservative. The robot will be unlikely to run into any real obstacle, however it may never reach some areas that a less conservative approach would allow. In the certainty grid approach this is implemented by interpreting the start value as closed area. Using an obstacle list it is not obvious how this could be accomplished. One might attempt to create the borders of a known reachable region. However, such a region would be a very complex object to represent as it would need to have internal borders, effectively making it a contour map of a two dimensional function with arbitrarily fine resolution.

A default state of unoccupied assumes that the sensor readings determine the location and boundaries of obstacles. This allows for more optimistic planning. It is a less conservative strategy since the algorithm may fail to place an obstacle, while not actually receiving readings that indicate the area is unoccupied. Implementing this on a certainty grid is also as easy as interpreting the default value as open area. The obstacle list approach automatically implements this assumption.

5.4.2 Prior Information

It is often the case that information about the area is known prior to the robot traversing the location. A map that allows one to easily incorporate this information is desirable. It should be noted, however, that given a simulator any map representation

can be preloaded, by simulating a run through the area on the known model.

Prior information can further be exploited by having the local map preload the less detailed information contained in the global map before starting its algorithm. This should allow the local algorithm to converge faster.

Both the certainty grid and the obstacle list can incorporate prior information without resorting to simulation. A certainty grid can be preloaded with values other than the default certainty. An obstacle list can start with a number of obstacles already part of the state. Depending on the type of filter used to incorporate new readings other information would also need to be provided for the obstacle list approach.

5.5 Delayed Evaluation

Many of the above requirements can be simultaneously accomodated by the addition of one extension to the certainty grid approach to mapping. Rather than keep a simple certainty value in each cell, one keeps information about what sensors have affected the cell. The certainty value then becomes a function of the state information kept in the cell. This allows for a more versatile strategy. One can have multiple evaluation functions which use different default values, resolve to different certainty values depending on the freshness of the data, and use different methods to account for prior information.

It is not obvious how this idea could be applied to an obstacle list. Instead one must create multiple obstacle lists, using a separate filter to update the state of each. Even this is less versatile than a certainty grid with delayed evaluation.

5.5.1 State Machine Abstraction

The use of delayed evaluation generalizes the idea of a certainty grid. Space is still partitioned into a grid, but instead of a simple certainty value, each cell contains a state machine. A function which accepts the current state of a cell defines the certainty for that cell. The information kept in each cell can be viewed as a history of all sensor readings that affect that cell. One can attempt to keep track of sensor hit order,

direction, or other information depending on the type of evaluation functions which will be used. If one knows that the family of evaluation functions is commutative with respect to sensor readings then a large number of sensor histories will map to the same state. Since this is the case with functions useful for a recursive approach, this can be used for global mapping without sacrificing a large amount of memory.

This abstraction makes a certain design choice explicit. In a recursive formulation it is typical to use the most accurate probability model possible for the sensor. In the case of a sonar this would involve adding probability on the cells at the edge of the cone and reducing probability inside the cone. The probability reduction inside the cone, however need not be uniform. Often closer areas will be reduced more than areas far from the sensor. This correlates to each sensor reading updating a large number of bits. One must make a choice between the number of bits used to differentiate spatial probabilities in one sensor reading versus keeping track of a larger number of readings with the same number of bits.

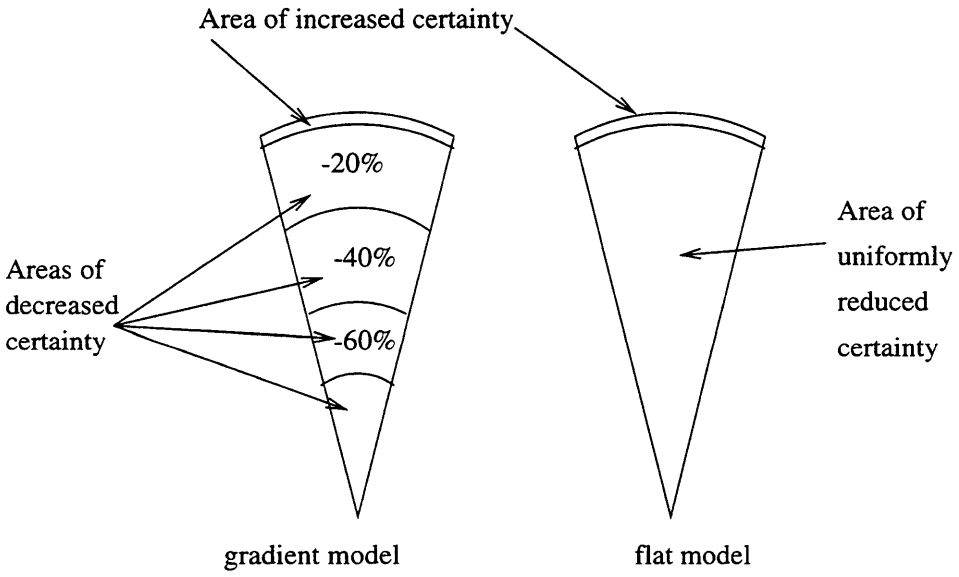


Figure 5-2: Various Sonar Update Rules

Two example sonar update rules are illustrated in Figure 5-2. The gradient method provides a more careful model of a single sonar return. It differentiates how much the certainty of an obstacle is changed by distance along the path of the beam. However,

it requires more state within each cell. The flat model provides a simpler model of a single sonar return. It only differentiates the interior and the edge of the return. This provides the advantage of reducing the state needed to track a single reading. In practical terms this means more state can be devoted to keeping multiple readings.

5.5.2 Advantages

Delayed evaluation allows for a number of powerful extensions. It is possible to have multiple planners running concurrently on the same map abstraction and interpret the state with different strategies. This is much more powerful than just using different certainty values as the cutoff point for obstacles. It is possible for one function to ignore a specific sensor and determine certainty values using only the other sensors. A precomputed certainty value has lost this information.

The use of delayed evaluation allows the same map to be used for both dynamic and recursive strategies. The set of states for each cell provides the domain of the possible evaluation functions. If the states for a typical recursive approach are differentiated by the last n readings then a dynamic strategy that uses those readings can also be applied to the state.

5.5.3 Disadvantages

One large disadvantage to this approach is that the certainty value must be evaluated during the critical query phase in the planner. This may place too large a time burden on a system that requires a low latency. If computing the certainty value takes significantly longer than reading a precomputed value it will likely have a great affect on planner performance.

Some evaluation strategies do not easily lend themselves to delayed evaluation. The state machine abstraction can allow for the implementation of any strategy, but some will require an impractical number of states. These strategies are typically those that attempt to order the sensor readings or have a complex spatial probability function. One must weigh usefulness of the complex update function versus the versatility

gained from delayed evaluation.

The last disadvantage is the amount of memory that may be used by a delayed evaluation scheme. The large amount of memory is already the main difficulty in implementing a certainty grid. If the state machine one uses has a large number of states then each cell may be very large compounding the memory problem. Oneway to deal with this is to sacrifice the size of the state. Another is to compile a very compact representation of the state machine which allows the state to be represented in a small number of bits.

5.6 Hierarchy

A useful tool for designing the map is to use a hierarchical representation. Heirachy can provide two important benefits. One is a savings in memory use and the other is an improvement in speed. A standard hierarchical representation for a flat map is a quadtree. This type of representation has a number of advantages. It allows for varying resolution and substantial savings in memory over a straight matrix. The idea is to partition the map into quadrants. Each quadrant has either information that applies to the whole quadrant uniformly or is itself partitioned.

The standard quadtree representation requires that the bounds on the map be known prior to its creation. It also suffers from a long lookup time compared to a straight matrix. Some of the operation time concerns can be ameliorated by clever use of cached pointers to lower areas of the tree for repeated operations. Another approach, however is to split each level into a larger matrix than 2x2. This reduces the depth of the tree and increases the benefit of caching pointers lower in the tree.

In the case of a certainty grid, one can reduce the amount of memory needed to store the map by using a hierarchical representation. It is also possible to create speed improvements by keeping information at each level of the hierarchy and applying sensor updates to the information at the highest level. This involves distributing the state representation of each cell along the path to the cell in the tree.

5.7 Handling Poor Sensor Readings

The map algorithm must be robust in the presence of poor sensor readings. The possible errors in individual sensor readings have been documented in earlier chapters. To deal with these one should attempt to preprocess the readings before incorporating them into the map. The use of the weighted Hough transform to reject bad laser readings is a good example of rejecting readings before they enter the map.

It should be recognized, however, that some bad readings will be added to the map. The certainty grid approach resolves this by modifying only small volumes of space and the use of a conservative recursive strategy to evaluate the certainty values. An obstacle list using an extended Kalman filter uses a dynamically updated covariance matrix to determine how much to weight a new reading versus the current state.

5.7.1 Latency

It is better to identify bad sensor readings early. To do this algorithms like the weighted Hough Transform, can be run on a set of readings, or the readings can be compared to earlier readings before incorporating them into the map. This approach causes sensor readings to be batched together before being incorporated into the map. One must therefore determine if the added robustness of the information is more valuable than the response time between physical sensor reading and incorporation into the map.

The decision to use a prefilter to toss bad readings depends greatly on the characteristics of each sensor. The laser is a good candidate to use a prefilter, because it produces a batch of readings in one sweep, and the time available to incorporate those readings is very small. The sonar does not present as clear a case. The main type of error one wants to protect against in a bad sonar reading is specular reflection. Protecting against this type of error is more difficult than identifying bad laser readings.

The amount of sensor batching needed to implement a weighted Hough Transform is naturally present in a single laser sweep. To deal with specular reflection with the

sonar one would need to batch sonar readings until either enough sonar readings or definitive laser readings were present to determine if specular reflection has occurred. This adversely affects the latency between physical readings and the incorporation of those readings into the map.

5.8 Chosen design

The design implemented on Companion is a variation of a certainty grid. There were a number of reasons for choosing the certainty grid over the obstacle list. The obstacle list is not as easy to modify as the certainty grid. Adding another sensor to an obstacle list implementation requires changing the update filter. Adding a sensor to a certainty grid implementation does not require changing the fusion routines of the sensors that are already present.

The obstacle list also lacks the versatility of the certainty grid. It is not apparent how one would implement the inherently conservative strategy of having space default to an obstacle. It is also not possible to implement delayed evaluation. To use multiple strategies one requires a separate map and a different filter for each strategy. It is also not possible to change strategies dynamically.

Finally the author is more comfortable implementing a certainty grid than designing an appropriate sensor filter. The certainty grid design is easier to understand, and more versatile. The main problem with the certainty grid design is its heavy use of memory. Dealing with the memory problem creates a number of unforeseen benefits. The next chapter will detail the implementation of the design.

The main ability lost in the choice of a certainty grid over an obstacle list approach is the improvement to the location estimate of the robot. However, the use of a certainty grid as the interface to the higher level planning does not preclude the use of an obstacle list and extended Kalman filter to help improve the estimate of the robot location. Currently the location estimate of the robot is done with a very simple algorithm which provides good estimates on a short term basis, but allows errors to grow unbounded. There are a number of improvements to this estimate

that can be made before one needs to change the method used in mapping the area. Incorporating an improved location estimate is considered the most important of the possible improvements to the robot.

Chapter 6

The Map Implementation

I don't care if you use a Kalman filter or a coffee filter. Just make it work! - Terence Chow

The choice of map implementation was strongly affected by concerns over speed and availability of the map to the planning system. The engineering decisions sacrifice resolution in areas where it doesn't matter for navigation and planning, in an attempt to allow common operations to go quickly.

The primary responsibility of the map is to determine the legal positions of the robot. The speed at which the navigation planner can query if a position is legal has a large affect on the speed and thereby performance of the overall system.

6.1 Certainty Grid Approach

The algorithm used in the *mapper* is based on a certainty grid. It was decided that the smallest cell area would be 1.5 cm square. The grid is effectively unbounded by allowing the addition of new high level cells. Each of the high level cells can contain as many as 10,000 of the lowest level cells.

The implementation allows the *mapper* to provide sensor fusion for planning operations requiring either a local map or a global map. It allows multiple processes to access the map and use delayed evaluation to dynamically determine the fusion strategy to use. It can provide resolutions of up to 1.5 cm and cover square kilometers

of area without needing to go to secondary storage.

6.2 Use of Hierarchy

The grid abstraction is implemented with a three level hierarchy. The level1 cells represent 1.5 x 1.5 meters. This level is not kept in a matrix, but rather in a hash table. The hash table starts empty and adds level1 cells when a sensor reading affects any part of the cell. By using a hash table at the top level, it is possible to extend the map in any direction without bound. The practical bound is defined by the length of the key used to access the hash table, however two four byte integers are used to identify each top level grid cell, so for practical purposes it is unbounded. The robot would need to travel over 3 million kilometers in one direction to fall off the map. Lack of memory would stop the robot long before the boundaries were reached.

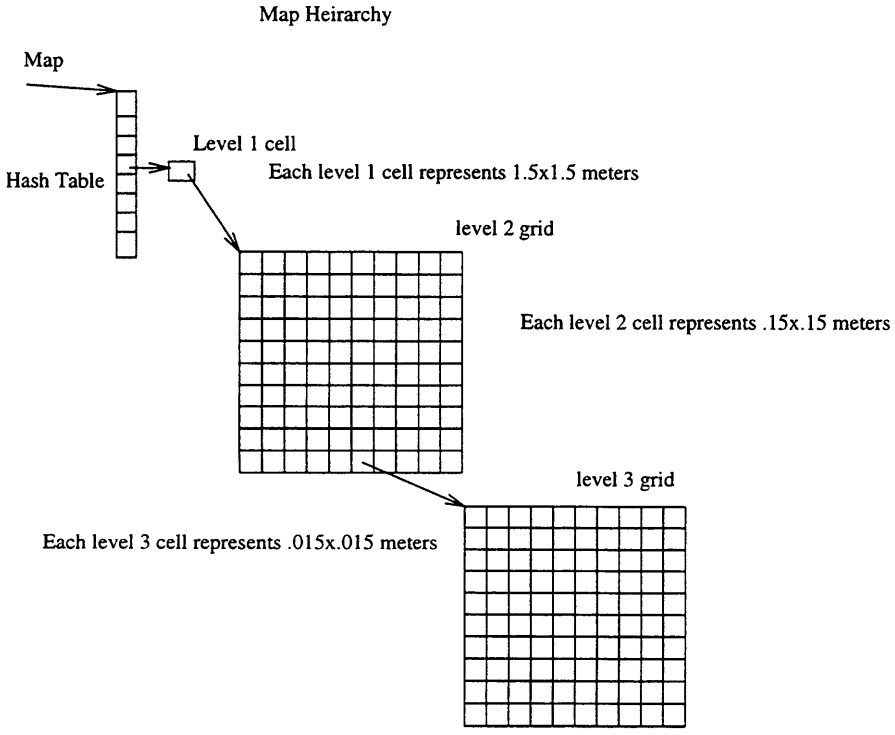


Figure 6-1: Map Cell Hierarchy

There are 100 level2 cells below each level1 cell. A level2 cell covers an area .15 x .15 meters. If the level1 cell is defined in the hash table, then a 10x10 level2 cell matrix is initialized. This means that every level1 cell defined in the hash table automatically causes 100 level2 cells to be created. This is done because no sensors on Companion give information that affects areas as large as 1.5 x 1.5 meters. Most of the memory used by the map falls in the lowest level, so this does not greatly affect memory use.

Below each level2 cell it is possible to create 100 level3 cells. Each of these cells covers an area .015 x .015 meters. These cells are not created by default. When a sensor reading hits only part of a level2 cell then the level3 cells below it are created. These cells are also created in a 10x10 matrix. Memory savings are obtained by the large percentage of level2 cells that do not have level3 cells defined below. This is possible if the entire level2 cell falls inside a sensor reading. It would be preferable if level3 cells were only created near obstacles. This however does not happen, as level3 cells are created in the middle of open areas when they fall along the edges of sonar returns. The solution to this problem is detailed later in the chapter.

The memory used by a level1 cell is 8 bytes, but this can be greatly increased without affecting the memory usage of the system as there are very few level1 cells. Each level2 cell uses 10 bytes. This size could be increased, as the total level2 memory usage is one tenth the maximum level3 memory usage. However, level3 cells do not approach the maximum usage, as they should not be defined in open areas. It is the size of the level2 cell that defines how large the global map can be. The level3 cell must be small, as a large number of them could be required to create the local map near the robot. Even though each level3 cell uses only 1 byte, there can be as many as 10,000 level3 cells below each level1 cell. For global mapping purposes it is questionable if one would wish to use 10K to describe a 1.5 x 1.5 meter area. Fortunately the level3 cells are not left in place for global mapping purposes. This idea will be explored in the memory management section.

The hierarchy does add pointer dereferences that would not be needed in a straight matrix implementation. However a space efficient implementation of a local map would require modular arithmetic on the grid indices to allow one to recenter the map with

robot movement. The hierarchical method does not need that modular arithmetic thus offseting the extra time needed for the pointer dereferences.

6.3 Sensor Counters

The state information kept for each sensor reading is a simple count. This allows the size of the state to be kept very small. The sonar is used to open areas, and the laser is used to identify obstacle edges. At the lowest level an 8 bit character is used to encode the number of laser hits, robot hits, and sonar hits. There is room for one robot hit, 4 laser hits, and 32 sonar hits. There are a number of options for dealing with overflow on the counters. The robot hit is simply set to one rather than incremented, as multiple instances of the robot occupying the same space gives little extra information. Both the laser and sonar increments the respective counter unless the counter is already at the max.

Later in the chapter it will be seen that with garbage collection this information will occasionally be thrown away allowing the map algorithm to start with new counters. In the absence of garbage collection it is also possible to implement a state machine. A simple example would be to trade 2 sonar readings for one laser reading. This reduces the counters whenever there are conflicting readings, allowing new readings to affect the state.

At the level2 cell there is more room to store information. Counts of all of the lower hits are kept as well as counters for readings which affect the entire cell area. Thus there are counters for both sonar and robot hits which affect the entire level2 cell. There is no similar counter for laser readings because a laser reading cannot affect an entire level2 cell. The state of a level3 cell is therefore defined as the sum of its local counters and the counters kept by its level2 parent. By distributing the state along the hierarchy a great deal of time can be saved in processing a sonar or robot return by incrementing one level2 counter rather than 100 level3 counters.

The level two cells keep two different types of counters which should not be confused. The above described sonar and robot counters count the number of sensor hits

which affect the entire cell replacing increments of the lower counters. There are two more counters which keep track of the total number of hits on the lower cells. The total sonar hits and total laser hits counters keep the sum of the effective counts of all the level3 cells in below the level2 cell.

6.4 Tables for the Sensor Fusion Function

The use of delayed evaluation improves the flexibility of the system. However, the speed of the operation is still of great concern. In the implementation on Companion it has been decided that the domain of the certainty evaluation function should be restricted to a small value. All of the information to be evaluated fits in a single character. This requirement is not very restrictive. It allows one to define a state machine of 256 cells with edges for each type of sensor update.

By restricting the domain of the evaluation function it is possible to precompute the function and store it in a table. This allows one the advantages of delayed evaluation for the price of one extra array dereference. It is still possible to efficiently implement delayed evaluation without seriously restricting the domain of the function. Using a technique called memoization, one caches the values of the function at common domain points (in a hash table) and looks first in the hash table before calculating a new point. It is also possible to simply make the evaluation function fast.

6.5 Message Passing

The *mapper* presents a cycle type interface to the planner processes. The mapping algorithm runs constantly and polls for requests from the planner processes. Unlike the cycle process, however, the *mapper* must also send commands to a number of blocked processes. In QNX this creates a messaging problem. It is possible for a response to a command to be misinterpreted as a request from a planner process.

To avoid this problem it is necessary to create a third process called the *mapper daemon*. The *mapper daemon* starts the *mapper* and begins as the only process which

can communicate requests to the *mapper*. Planner processes which wish to communicate to the *mapper* send a request to the *mapper daemon*. The *mapper daemon* sends a command to the *mapper* telling it to accept requests from that planner process. The *mapper* is thus able to keep a list of all the processes from which it will accept commands. This solves the message communication problem, and is implemented with the same interface as the cycle process. The *mapper daemon* is transparent to the programmer creating a planner process.

6.6 Shared Memory

The main data structure must be kept in shared memory to provide the planning programs an efficient implementation of the robot safety query. QNX provides for simple implementation of message passing, however, the basic query will be called too often to allow it to be implemented as a remote procedure call. Given this constraint, the data structure must be shared between the *mapper* and the planner processes that call the basic robot location query. The use of shared memory adds the constraint that the *mapper* and planner processes must reside on the same processor. This reduces the ability to distribute the system over multiple nodes, but is worthwhile given the possible performance improvement.

The planner does not need to alter the map to make use of the data. However, it may need to request that the *mapper* not make any changes while it is using the map. The planning system designed for Companion includes two processes that make use of the shared memory map. Both require that the map does not alter while the planning is in progress.

6.6.1 Address Independence

There is a subtle requirement that greatly complicates the implementation of the map in shared memory. Unless the base address of the shared memory segment is the same in all processes referencing the map, pointers will not work. The QNX operating system does not allow a process to request the base address to use when attaching

a shared memory segment. It is therefore necessary to implement an alternative to pointers in the data structure.

This was accomplished by having each process which attaches the shared memory segment keep a structure containing the base address at which the shared memory starts and standard offsets into that memory. All pointer references are then replaced with array indices referenced from local offsets. To implement this a smalloc() (shared memory malloc function) was created along with an sfree(). The memory management routines can be created with access to the implementation of these functions.

6.6.2 Locking

It is necessary to implement some type of locking to allow the planner processes to read the map without the *mapper* altering it. The data structure can either be locked as a whole, or have locks on smaller portions of the structure. Locking the whole data structure is easily implemented by message passing to the *mapper*. However, a more versatile method is to lock each level1 cell independently. This allows multiple processes to alter the map at the same time. There are, however, a number of tricky interactions which must be resolved to avoid deadlock or a violation of the planner requirements.

The planner processes do not know which level1 cells will need to be locked to complete a search. It is therefore better from the point of view of a someone implementing a planner process to lock the whole map during a search. This is easy to implement via message passing. The mapper's message interface includes a request to freeze operations on the map, and another to release the lock. The mapper can still process readings during a map freeze. The requirement is that all other processes see the map frozen at the state just before a freeze request. The mapper can update copies of the level1 trees and replace them in the hash table when the map is unfrozen. This allows the planner processes to see a stable map while not seriously affecting the ability of the *mapper* to continue processing.

The above strategy works well without needing to lock a level1 cell. However if one wishes to have multiple processes write to the map then it is better to have the

ability to lock a smaller portion. Adding a garbage collector to the system creates this problem. The garbage collector will need to modify the map by removing level3 cells and consolidating the information in the level2 parent. To do this it must make sure that the *mapper* process is not modifying the cell and no other process is reading the cell. It is possible to avoid write conflicts by allowing a write lock on the whole tree below a level1 cell. The *mapper* and garbage collector then obtain write locks on the level1 cell before attempting to modify it. One then avoids deadlocks by simply restricting each process to hold only one write lock at a time, and make sure the processes do not block before releasing the lock. Each process can hold a write lock and modify a copy of the level1 tree while the read lock is held by a planner process. When the read lock is released then each can update the map. However, if one does this then one must again make a careful analysis to show that the implementation can not cause a deadlock since the process must block until the read lock is lifted.

Having more than one process able to modify the map complicates the read lock mechanism. All processes that can modify the map need access to the state of the read lock. This can be accomplished on QNX, but is not as simple as the message passing method. The *mapper* process can post the read lock to the system by defining a privileged name. It must then make sure that the memory manager has completed its last operation before returning the read lock confirmation to the requesting process.

6.7 Memory Management

As was noted in the architecture section, the main battle in implementing a certainty grid is to keep the memory usage to a reasonable level. It can be useful to implement garbage collection. As described so far the map never removes any pointers so garbage collection is not needed. However it is the case that a large number of level3 cells are created not because the sensor readings fall next to an object, but instead because the edge of the sensor reading falls across some cell that is in open space. Removing these redundant level3 cells and consolidating the information back up to the parent level2 can save a great deal of memory.

Along the edges of obstacles it can also be useful to consolidate information back up a level, as this will help implement more of a dynamic stability approach for local mapping. This is also necessary as the limited amount of state kept in the level3 cells, will keep further readings from having any affect.

6.7.1 Garbage Collection

Garbage collection should focus on collecting memory that is no longer being well used. This means level3 cells that are defined far away from the robot or are very old. The amount of drift error and the possibility of moving obstacles make the details built up in old level3 cells questionable. The information in those cells is already consolidated in the parent level2 cell in the form of sensor totals. All that is necessary is to return the memory used for the level3 cells and null the pointer to the lower matrix in the level2 cell.

It is also possible that during this operation one wishes to relax the certainty value in the level2 cell back toward uncertainty. This would allow for a better dynamic stability implementation. If global mapping is the goal then that should not be done. Given the use of delayed evaluation and the larger state available in a level2 cell it is possible to implement both.

6.8 Sensor Utilization

The control of the commanded sensors rests with the *mapper* process. The strategy implemented is a very simple passive scan strategy. The laser makes sweeps at a set elevation, and the sonar fire in a specific order. This strategy is sufficient to create a map, but can definitely be improved upon.

Using only information within the *mapper* it is possible to use the laser to attempt to verify the sonar readings, thus actively attempting to account for sonar reflection. This would also tend to cause the laser to hit more obstacles. Note the laser is only adding significant information by hitting obstacles, it does not provide enough converge to be useful in the negative sense of indicating the absence of obstacles. It is

also possible to implement more complex strategies involving the use of the planner to direct the sensors to the areas that the planner most needs accurate information. This type of strategy was not implemented on Companion, and is beyond the scope of this thesis.

Chapter 7

Map Realization on Companion

7.1 What was Implemented

The design outlined in this thesis is being implemented on the Companion mobile robot. All of the modules shown in Figure 2-3 have been implemented. The quality of the implementation varies. The earliest modules are very reliable and robust. These include the *sonar*, *cycle*, *sound*, and *trajectory* processes. The latter modules, including the *mapper* are in a prototype stage.

The *laser* process works well in simple tests, but experiences degraded performance during the operation of the full system. A number of improvements to better account for the time delays and movement of the robot would greatly increase the performance of the laser module. The mapping results that have been obtained are impressive considering the relatively poor performance of this key system.

Delayed evaluation has been implemented. This gives the planner the ability to dynamically change sensor fusion strategies. It also allows the developers to test a large variety of fusion strategies offline. The limited amount of state kept at the lowest level (8 bits) is the main limit on the versatility of the implementation. Each of the test series use the bits to to count different sensor activities. The *a* series counts sonar, laser, and robot hits. The *c* series does not track the robot hits. Instead it uses the bits to implement a better sonar model. The *b* series was a failed attempt to implement what was correctly done in the *c* series.

Garbage collection of the outdated local map cells has not been implemented. This feature is considered necessary for an implementation which runs for extended periods of time. It is, however, unnecessary for the purposes of testing the rest of the design and the mapping performance of the current system.

A simple sensor fusion strategy is used in the implementation. This strategy uses a flat sonar model, which has a constant probability across the interior of the cone. While this may be a step backward in terms of fidelity, it allows for a much more compact representation of the state information. It was decided that the use of delayed evaluation and the reduction in the size of the state was worth the reduction in sonar fidelity. This choice is not fundamental to the system, a Bayesian probability method for fusing the readings could be integrated into the system while maintaining the use of delayed evaluation. The performance which has been obtained, however, argues for further effort to be expended elsewhere.

7.2 Tests

A number of tests were performed with the current implementation of the *mapper*. Three representative tests will be presented here. A number of the tests which are not presented were flawed due to errors which were not directly related to the *mapper* process. Tests $a1$ through $a5$ were flawed due to programming a path which caused Companion to run into walls. This required the operators to stop the robot manually before it completed the pattern as well as corrupting the readings. The entire b series of tests were flawed, because the modification to the *mapper* to track the edges of the sonar beams was incorrect. The tests which are presented were done at least twice to verify repeatability.

All of the tests presented involve running the robot in a preplanned path through the lab. The safety mechanisms and planning were not activated. This procedure is implemented by creating a scripted path for the robot to follow and commanding the *trajectory* process to follow that script. This seemingly simple procedure was more difficult than anticipated. The input to the *trajectory* process is tailored for the output

of the path planning algorithms, it is not very intuitive for human control. Test runs $a1$ through $a5$ were corrupted by the need to enter the room to stop the robot from hitting a wall due to an incorrect path being given. Tests $a6$ through $a10$ worked well, only requiring the operator to be in the vicinity of the robot during the start.

The series a tests attempted to have the robot circle the room in a space filling pattern, as shown in Figure 7-1. The series c tests were done in a slightly modified room. A simple repeated path, moving back and forth along the axis of the room was used after the experience of designing more complex paths for the series a tests. The other important variables in each test include the room setup, the sensor utilization strategy, and the mapping strategy.

All of the series a tests were done in a single room setup shown in Figure 7-1. The series c tests were done in the same laboratory, but with a slight modification shown in Figure 7-2. The series a tests used laser sweeps which scanned 100 degrees of azimuth centered on the forward direction of the robot. The series c tests used laser sweeps that covered 330 degrees of azimuth and took 100 readings per sweep. The series a tests used the sonar only to clear area in the maps, the edges of the sonar returns were not marked. The series c tests did mark the edges of the sonar returns but otherwise still used a flat sonar model for the interior of the cone.

7.3 Results

The performance of the low level systems greatly affects the possible performance of the *mapper*. As expected the sonar system was more reliable than the laser. Figure 7-3 shows the map created by raw sonar ranges in the $c1$ test. For comparison Figure 7-4 shows the areas hit by raw laser returns on the same test. All test results use a scale based on the map cell. Each unit corresponds to 15 cm. The laser has a large number of completely spurious readings. The raw sonar readings make a tolerable map without a large amount of sensor fusion. The disappointing results of the raw laser indicates the need for further work on improving this sensor.

The performance of the fusion algorithm using a flat sonar model marking the edge

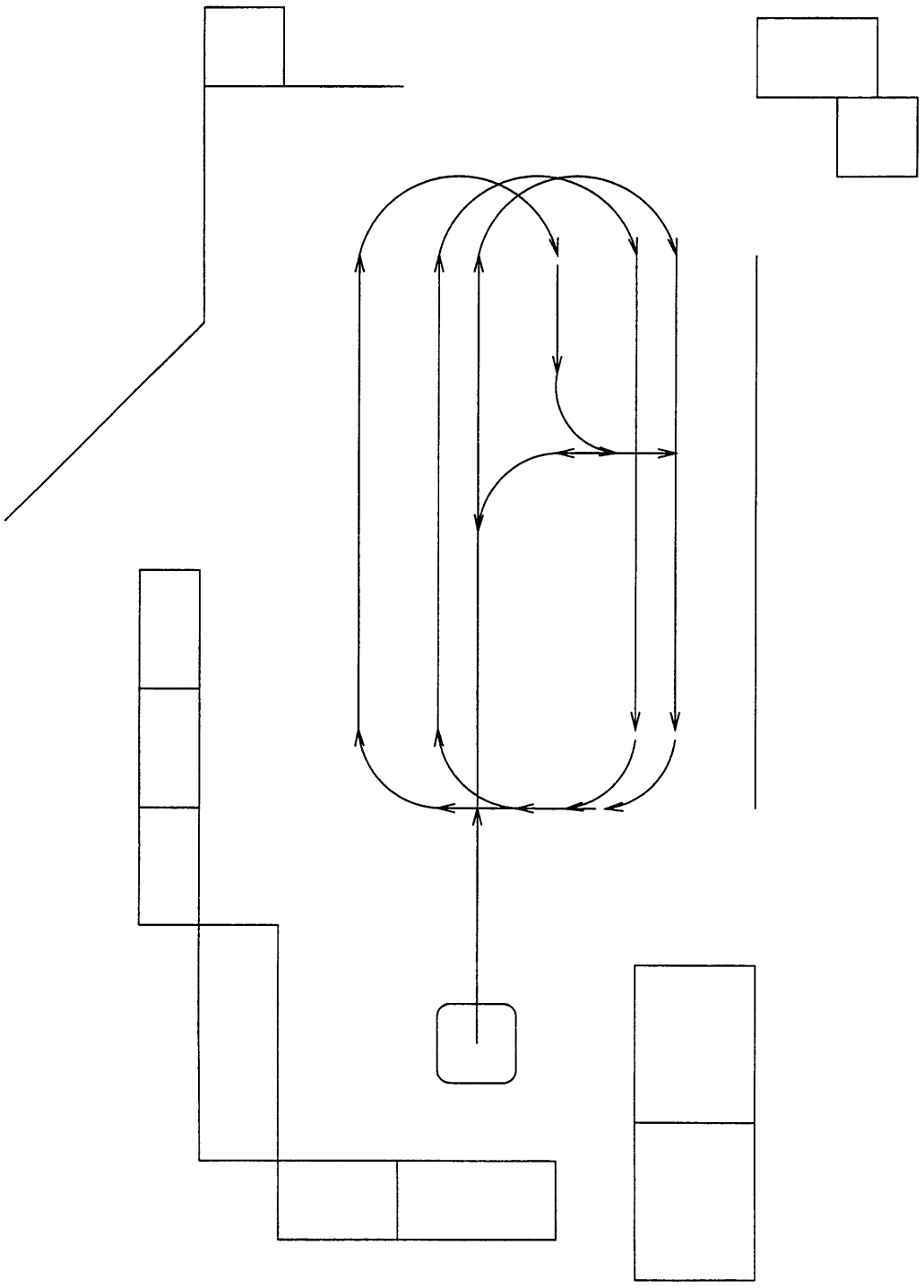


Figure 7-1: Testing area: test series *a* movement pattern

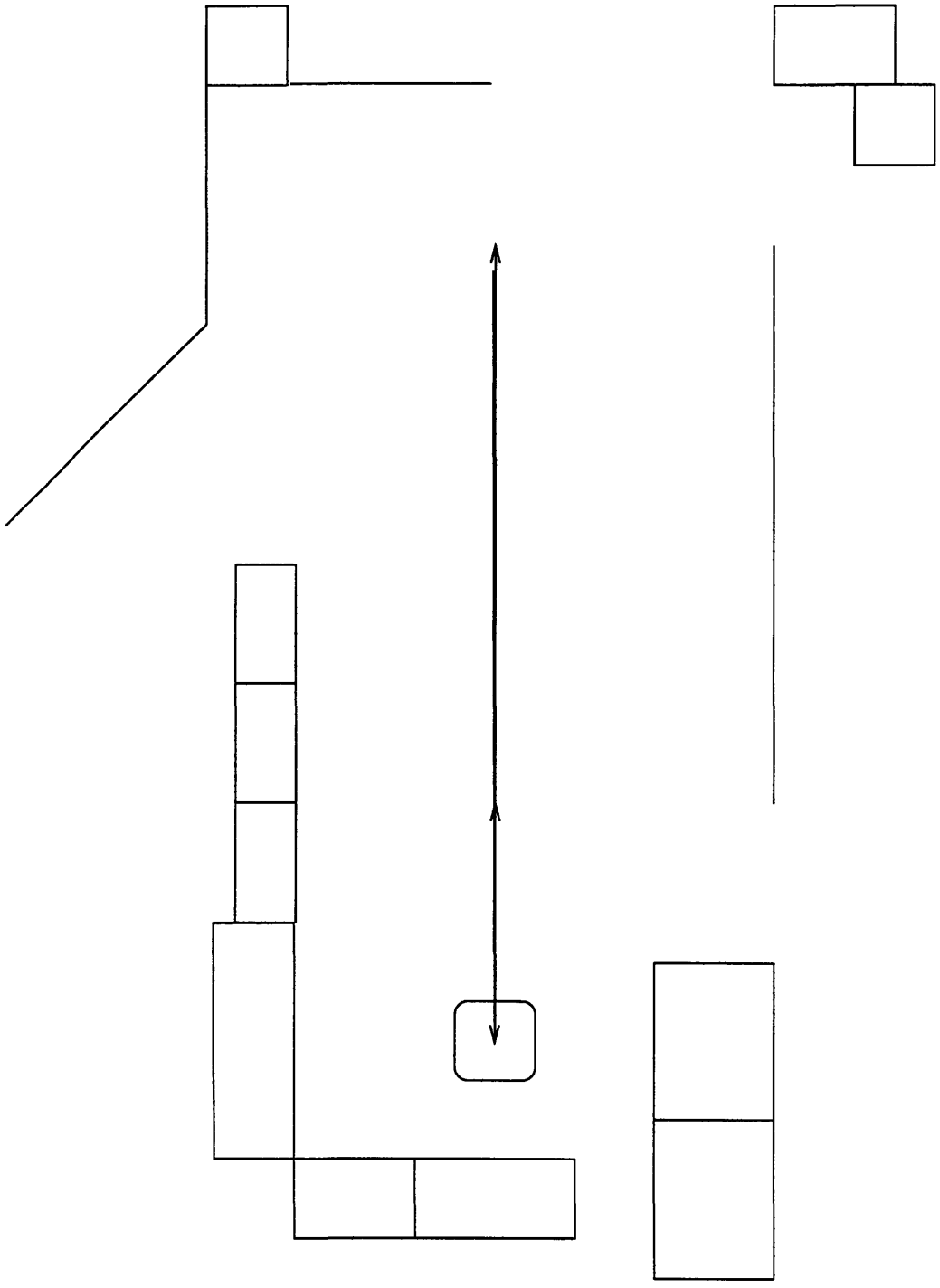


Figure 7-2: Testing area: test series *c* movement pattern

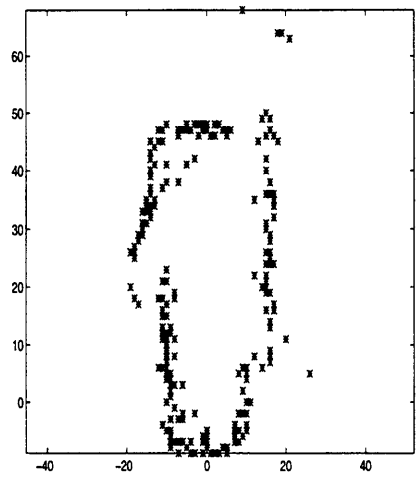


Figure 7-3: Sonar only map: test c1

of the beam, and the use of the laser is shown in Figure 7-5. This map is created by a simple linear combination of the number of sonar readings, lasers readings, and sonar edge readings compared to a constant. This is a function which is feasible to implement and run in realtime with the use of delayed evaluation. The performance of this function is robust to a large range of values. The actual binary function used to create the map is $-1 * s + 340 * l + 100 * e > 1$. The large laser values used are due to the fact that the saved maps are global maps. This means that each sonar hit is represented by 100 lower level sonar counts. The laser is here weighted at 3.4. This means 3.4 sonar hits wipe out a laser reading. This type of function is a very simple dynamic stability approach, which seems to have relatively good results across multiple runs.

Figure 7-6 shows the composite maps created from two tests of the *a* series. These maps use the binary function $-1 * s + 340 * l > 1$. The maps created by the composite information can be seen to accurately represent the room. The composite function compensates for the spurious laser readings from the interior room. The laser returns enough correct readings of the walls to compensate for the specular reflection readings

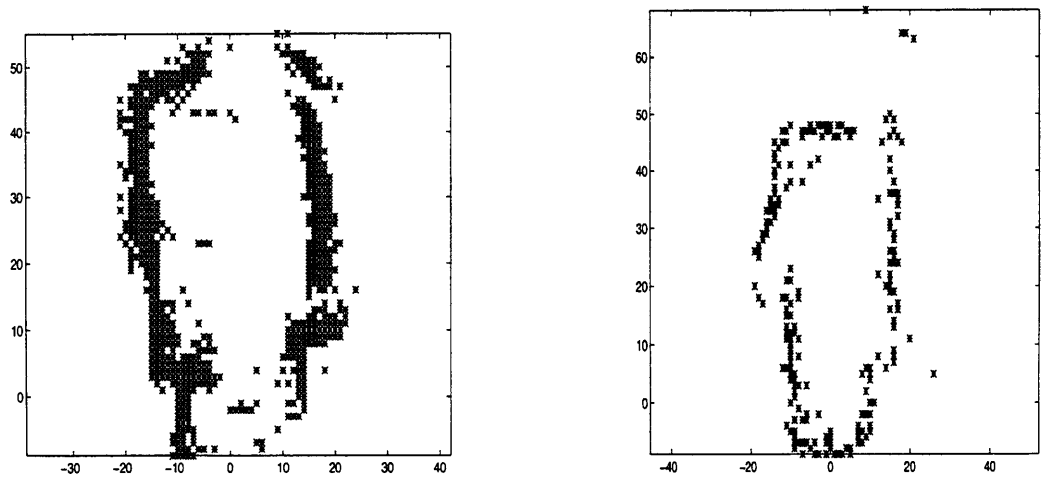


Figure 7-4: Raw Laser Hits

of the sonar. Together the two sensors provide a good method to alternatively remove and place obstacles in the map. If the laser system were improved it is conceivable that a well tuned dynamic system implemented could allow for moving obstacles.

The search algorithm does not require the solidity of the walls shown in these maps. It is therefore possible to reduce the weight of the laser without seriously affecting planning performance. The reduction in the weight given to laser information would reduce the chance of spurious readings creating false obstacles. Considering the number of stray readings from the laser, this may be a reasonable approach.

The contour maps shown in Figure 7-7 show that the data collected in the series *a* tests agrees on room shape and location. The contour plots help to present the information in a form that is easier to see. No claim is made that the contour plots are feasible to implement in realtime for the search process.

The results shown were obtained with an implementation that is able to run very quickly due to a number of simplifications of the sonar model. It is possible to create a higher fidelity sonar model and improve the performance. However, it is not clear that the present performance of the *mapper* in locating obstacles is inadequate. The

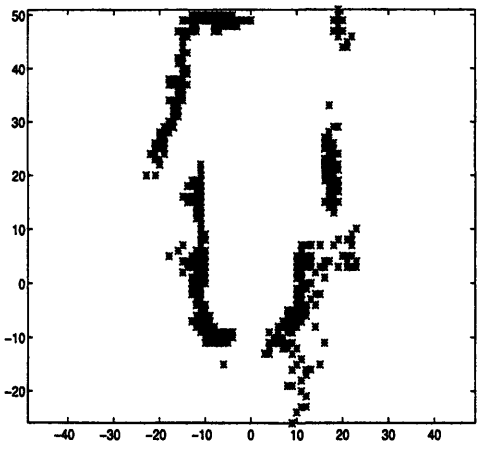


Figure 7-5: Composite map; test c1

improvement of the sonar model would improve the robustness and accuracy of the maps, but might affect performance in other ways. This speculative improvement is not a high priority. Improving the laser process is something that has been clearly shown to be necessary. This is especially important if a feature based relocalization routine is to be added.

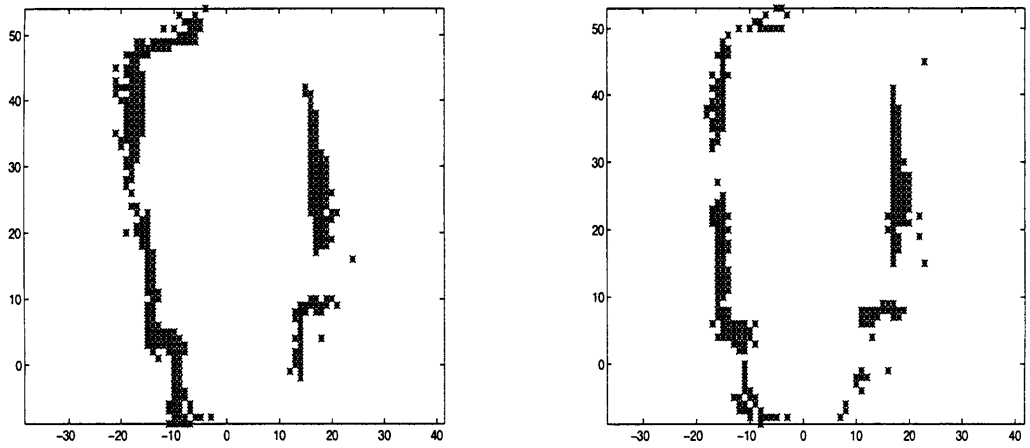
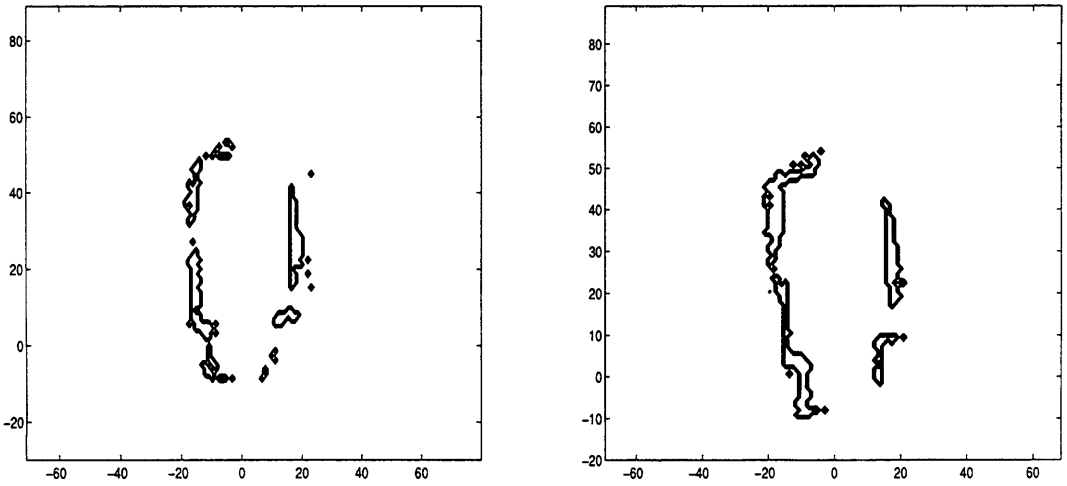
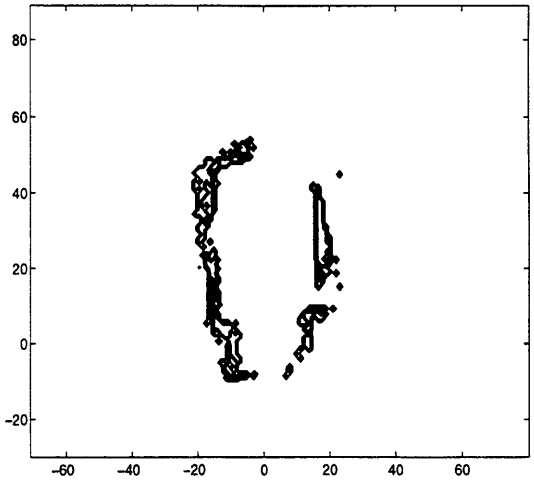


Figure 7-6: Composite maps



Test 6

Test 7



Overlap of test 6 and test 7

Figure 7-7: Contour Maps created with the map data from tests 6 and 7

Chapter 8

Conclusion

8.1 Effectiveness of Design

As with any large project, it is at the end that one sees what should have been done all along. Companion is a very complex system, yet the number and severity of the flaws is smaller than might be expected. The overall architecture of the robotic system is well designed. It is adaptable and efficient in its implementation. Unsurprisingly, the lowest levels of the system are the most reliable and robust. Those modules were the first completed and have been continuously debugged and upgraded since that time. The highest levels of the system are adequate, yet there remains much that could be done to improve the performance and reliability of the platform.

The strength of the *mapper* module is its ability to dynamically create a map useful for both global and local mapping. The main flaw of the module is the failure to bound the error accumulated in dead reckoning. Adding a beacon system would provide a simple solution to this problem. A much better solution, however, would be the addition of a feature based map which updates the dead reckoning estimate by recognizing landmarks. Such a system using walls as landmarks has been developed by Larsson, Forsberg, and Wernersson [6]. The combination of the *mapper* created in this thesis and an efficient implementation of of Larsson's work, focused only on keeping track of enough landmarks to relocalize should provide a feasible real time implementation. This design provides a bound on the error in the dead reckoning

estimate of position and removes the need to make serious assumptions about the types of obstacles that will be encountered.

Without the feature based map, the current system will work, but not in as pleasing a manner. Over time, drift due to dead reckoning error basically amounts to a translation and rotation of the coordinate system. The older parts of the map degrade as new sensor information updates the map with the correct information. This ability to provide good relative obstacle location despite poor absolute location information accounts for a great deal of the reliability in using this mapping system for local planning.

8.2 Contributions

The *mapper* module created in this thesis provides a number of advantages over previous mapping routines. The use of delayed evaluation greatly improves the versatility of the system. Multiple processes can use the map for a wide variety of purposes. Each process can chose a different strategy for interpreting the map, and ignore or weight sensors differently. The delayed evaluation feature is used in the current system, allowing a single map to provide information for two different search processes. Implementing the feature with a finite state machine allows one to choose precisely what information is kept in each sensor reading. It also highlights the tradeoffs between space, computation time, and completeness of the model.

The decoupling of the map from the robot also improves the efficiency of the map implementation. Previous certainty grid maps have maintained the robot at the center of a square map. The ability to have nonsquare maps and to locate the robot anywhere on the map greatly improves the performance of the implementation in terms of both space and time. The hierarchical structure used in the map improves on earlier map implementations in terms of both space and time. The use of hierarchy also provides a practical method for recovering memory from little used portions of the map. This design feature allows for practical long term operation. The data structure uses memory efficiently enough to be able to maintain the entire map in in

main memory. Should very large maps become necessary it is also possible to use the garbage collection routine to store the information to disk as it cleans.

8.3 Future Work

Companion has many areas open for future work. The first improvement to the map should be the addition of a feature based routine to help localize the robot. This would be simplest to add as another operation in the *mapper* process, as the sensors are controlled from that process. The *cycle* process already has an interface provided to allow for corrections to its estimate of the position and heading. The advantages of adding this capability extend beyond the performance of Companion. Certainty grids have been used in a number of applications. The additional benefit of tracking the location of the vehicle with high accuracy without the use of beacons would improve a number of those applications.

The control system for Companion is versatile enough to be used on a variety of robots. It is conceivable that the Companion architecture could be used on a ground station to control cheaper vehicles. A number of such vehicles are currently being built in the lab and will require sophisticated control. While the robots do not have the computational capability of Companion, they do have a ground station which must take over most of the higher level functions.

The other areas of exploration for improving the mapping and planning system follow the suggestions of Elfes [5]. Integrating the mapping and planning systems to allow for more intelligent sensing should improve performance. However, this is not the path I would recommend for further work on Companion. The addition of a feature based map improves an identified deficiency. Beyond that deficiency it is not yet apparent that the mapping is the area in need of the most improvement. On Companion the *trajectory* system currently uses a simple control algorithm for path following. Improving that control algorithm would greatly improve performance. The current dead reckoning can also be improved by the addition of a Kalman filter to better track the vehicle state.

Finally the area in most need of improvement for Companion is the user interface or ground station. The Companion platform is a superb platform for doing experimental work on robots. The architecture which has been designed and implemented provides a versatile and powerful interface for the systems developer. It does not however provide a simple interface for someone who is not a developer on the system. This is an area that requires a different type of expertise and should not be ignored if practical applications of the system are intended.

The map architecture described in this thesis provides a number of benefits which should be continued in future systems. The complete system implemented on Companion represents a great value. It provides a powerful platform for the design and testing of new robotic system concepts. The potential inherent in the vehicle has not yet been explored. Hopefully this thesis, Terence Chow's thesis, and the current system implemented on Companion will provide a foundation for continued work by those who remain in the Draper Unmanned Vehicle Laboratory.

Bibliography

[1] Johann Borenstein. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.

[2] Stephen Borthwick and Hugh Durrant-Whyte. Dynamic localisation of automomous guided vehicles. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 92–97, October 1994.

[3] Terence Chow. Software architecture, path planning, and implementation for an autonomous robot. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, June 1996.

[4] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[5] Alberto Elfes. Dynamic control of robot perception using stochastic spatial models. In *Proceedings of the International Workshop on Information Processing in Mobile Robots*, pages 203–218, March 1991.

[6] Ulf Larsson Johan Forsberg and Ake Wernersson. On robot navigation using identical landmarks: integrating measurements from a time-of-flight laser. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 17–26, October 1994.

[7] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. In *NATO ASI Series*, volume F52, pages 253–276, 1989.

[8] W. K. Stewart. *Multisensor Modeling underwater with Uncertain Information.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, June 1988.