

2001

Mapping by Cooperative Mobile Robots.

Thomas Oliver Smailus
Louisiana State University and Agricultural & Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_disstheses

Recommended Citation

Smailus, Thomas Oliver, "Mapping by Cooperative Mobile Robots." (2001). *LSU Historical Dissertations and Theses*. 364.

https://digitalcommons.lsu.edu/gradschool_disstheses/364

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Historical Dissertations and Theses by an authorized administrator of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

MAPPING BY COOPERATIVE MOBILE ROBOTS

A Dissertation

**Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy**

in

The Department of Computer Science

by

Thomas Oliver Smailus

B.S., Louisiana State University, 1990

M.S., Louisiana State University, 1992

August, 2001

UMI Number: 3021453

**Copyright 2001 by
Smailus, Thomas Oliver**

All rights reserved.

UMI[®]

UMI Microform 3021453

Copyright 2001 by Bell & Howell Information and Learning Company.

**All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.**

**Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346**

**© Copyright 2001
Thomas Oliver Smailus
All Rights Reserved**

Acknowledgements

I am eternally grateful to the faculty and staff of the Louisiana State University departments of Computer Science and Electrical and Computer Engineering for the guidance and opportunities they have given me over the course of my fifteen years here at the University obtaining my undergraduate and graduate degrees. Dr. S. S. Iyengar has been a great inspiration for the future that is possible and has provided the guidance and some of the tools necessary to shape that future. Dr. J. Trahan has always been a solid reviewer of my progress and a motivation to do more.

I am also grateful to the TI computer I stumbled upon in the store in 1982, which started my interest with all things computer. I must thank Mr. T. Kalla for showing me what else I could do with a computer in 1983 and beyond and for giving me the freedom to explore well beyond the confines of the course curriculum.

I would like to thank Atari® and Commodore® for showing us what we could do with this new technology. Atari even helped me write a term paper as an undergraduate in 1989 at LSU; Long Live AtariWriter® (though it had to be stored in more than one file as a result of the 48K byte memory of the Atari). I would also like to thank IBM® for having half of an idea with the PC and then screwing it all up and Microsoft® for taking anything that seems useful and reinventing it in their image but with too many bugs to make it work well and too few bugs to start a world revolt. I know because I've written this document with such software.

Finally I would like to thank my uncle, Bernd, and grandfather, Jacob, who provided no small amount of inspiration and my parents and wife, Melanie, who continually believed in me and urged me to continue with this effort as it dragged on through the 8 years it has consumed. And to my father-in-law, Oscar, I say this, "Ha, I'm done with it!"

Now the real fun begins.

Table of Contents

Acknowledgements	iii
List of Tables	vi
List of Figures	vii
Abstract	x
1. Introduction	1
1.1 The Application of Mobile Robots to Mapmaking.....	1
1.2 Modeling the Environment.....	3
1.2.1 Grid-Based Mapping.....	8
1.2.2 Graph-Based Mapping.....	13
1.2.3 Combined Grid-Graph-Based Mapping.....	17
1.3 Imperfections in Mapping.....	17
1.3.1 Two Forms of Error.....	19
1.4 Cooperation in Mapping.....	23
1.5 Scope of this Dissertation.....	24
1.6 Organization of this Dissertation.....	25
2. Existing Methods of Robotic Mapping	26
2.1 Methods of Robot Map Construction.....	26
2.2 Error Detection in Agent Self-Location.....	38
3. The Problem and the New Method	45
3.1 The Problem of Cooperative Mapping Using Agents.....	45
3.1.1 A Cooperative Agent.....	47
3.1.2 Modeling Error in Simulations.....	49
3.2 Precision Mapping.....	50
3.2.1 Error Detection in Self-Location Data.....	51
3.2.2 A Hybrid Environment Model.....	52
3.2.3 Error Correction.....	54
3.2.4 Error Containment.....	56
3.3 Distributed and Cooperative Solutions.....	59
3.3.1 Concerns About the Singh-Fujimura Algorithm.....	62
3.4 A Proposed Mapping Architecture.....	64
4. Map Description Language – A New Paradigm for Robot Mapping	74
4.1 Introduction.....	74
4.2 Time Element.....	76
4.3 Position Element.....	77
4.4 Action/Reaction Element.....	80
4.5 Brush Element.....	81
4.6 Correction Factor Element.....	82
4.7 Error Detection and Correction in Historical Data.....	84
4.8 Cooperative and Distributed Map Building.....	90
4.9 An Example Illustration of the MDL Paradigm.....	92
5. Experimental Simulation of MDL and Map Correction	100
5.1 Conditions of the Simulation.....	100
5.1.1 World Model.....	100

5.1.2 Implementations and Systems.....	100
5.2 Simulated Mapping of Objects	107
5.2.1 Experiment One.....	107
5.2.2 Experiment Two	111
5.3 Analysis and Interpretation of the Results	113
5.3.1 Experiment One.....	113
5.3.2 Experiment Two	121
5.3.3 Synopsis of Results.....	130
6. Conclusion and Future Work.....	133
6.1 Contribution of this Work	133
6.2 Future Directions of this Research	134
6.3 Final Words	136
Bibliography	137
Vita.....	142

List of Tables

Table 1. Brief Overview of Mapping Research	9
Table 2. Summary of Grid-based Approaches	35
Table 3. Summary of Graph-based Approaches	36
Table 4. Classes of Error Detection	40
Table 5. Single Agent Object Trace Algorithm	69
Table 6. Multi-agent Space-fill Algorithm	71
Table 7. Effect of Error on Distance Traveled.....	86
Table 8. Application of Time Dependent Transform	97
Table 9. Rectangle Results	115
Table 10. Ellipsoid Results.....	116
Table 11. Triangle Results.....	117
Table 12. 2 Robot, 3 Object Results.....	126
Table 13. TDT Performance Results	127

List of Figures

Figure 1. Grid Based Mapping	4
Figure 2. Graph Based Mapping.....	6
Figure 3. Effect of Robot Movement.....	7
Figure 4. Ideal Occupancy Grid Sensor	11
Figure 5. Leonard Map Building Algorithm.....	15
Figure 6. Error Accumulation: Growth of the Uncertainty Sphere.....	21
Figure 7. Mapping Drift.....	22
Figure 8. System Overview.....	46
Figure 9. Position and Error Produce an Uncertainty Sphere.....	48
Figure 10. Hybrid Map Representation	53
Figure 11. Map Data Merging	58
Figure 12. Proposed Mapping Architecture within a Single Mapping Agent.	66
Figure 13. Focus Subset of Architecture	68
Figure 14. Simple and Complex MDL	74
Figure 15. Overview of the MDL Mapping Technique.....	76
Figure 16. Robot Configurations.....	78
Figure 17. Reconfigurable Robot Developed by JPL, CalTech [Schenker-00][Schenker-01]	79
Figure 18. Another Small Reconfigurable Robot Developed by JPL, CalTech [Wilcox-96].....	80
Figure 19. Brush Shapes	82
Figure 20. The Effect of Systematic Error.....	86
Figure 21. Linear and Nonlinear TDT	88
Figure 22. Staircase TDT Applies Correction More Intelligently.	89
Figure 23. Distance Scaled Staircase TDT.....	90
Figure 24. Map Data Isolation	91
Figure 25. A Detailed Example, Part 1	95

Figure 26. A Detailed Example, Part 2.....	96
Figure 27. Corrected Map for Example.....	98
Figure 28. Gaussian Distribution Function.....	104
Figure 29. Gaussian Based Density Function for Error Simulation	104
Figure 30. UROSYS Desktop.....	106
Figure 31. UROSYS Simulator Configuration.....	106
Figure 32. UROSYS Error and Robot Model Configuration	106
Figure 33. Shapes Mapped	108
Figure 34. Obstacle Perimeter Tracing.....	110
Figure 35. Multi-object Environment for Cooperative Mapping.....	112
Figure 36. Simulator GUI Workspace with Two Agents Mapping	112
Figure 37. Rectangle Results	115
Figure 38. Ellipsoid Results.....	116
Figure 39. Triangle Results	117
Figure 40. Absolute Gain in Ellipsoid	118
Figure 41. Absolute Gain in Triangle	118
Figure 42. Absolute Gain in Rectangle.....	118
Figure 43. Normalized Gain in Rectangle	119
Figure 44. Normalized Gain for Ellipsoid	119
Figure 45. Normalized Gain for Triangle	119
Figure 46. Completed Global Map and Global Coverage.....	123
Figure 47. Uncorrected and Corrected Global Map.....	123
Figure 48. Environment Model.....	123
Figure 49. Coverage Gains from Linear TDT	125
Figure 50. Coverage Gains from Stepped TDT	125
Figure 51. Coverage Gains from Scaled Stepped TDT	125
Figure 52. Results with 2 Robots and 3 Objects.....	126

Figure 53. TDT Performance by Type.....	127
Figure 54. Normalized Results with 2 Robots and 3 Objects.....	128
Figure 55. Normalized Gain Over Traction Loss Factor	128

Abstract

Constructing a system of intelligent robotic mapping agents that can function in an unstructured and unknown environment is a challenging task. With the exploration of our solar system as well as our own planet requiring more robust mapping agents, and with the drastic drop in the price of technology versus the gains in performance, robotic mapping is becoming a focus of research like never before. Efforts are underway to send mobile robots to map bodies within our solar system. While much of the research in robotic map construction has been focused on building maps used by the robotic agents themselves, very little has been done in building maps usable by humans. And yet it is the human that drives the need for mapping solutions.

We propose a computational framework for building mobile robotic mapping systems to be deployed in unknown environments. This is the first work known to address the general problem of mapping in unknown terrain under the effect of error in readings, operations and systems that employs more than a single robot. The system draws upon the strengths from research in various robotic related areas by selecting those components and ideas that show promise when applied to mapping for human reading via a distributed network of heterogeneous mobile robots. This application of multiple mobile robots and the application to human end-users is a new direction in robotics research. We also propose and develop a new paradigm for storing mapping-agent generated data in a way that allows rapid map construction and correction to compensate for detected errors. We experimentally test the paradigm on a simulated robotic environment and analyze the results and show that there is a definite gain from correction, particularly in error rich environments. We also develop methods by which to apply corrections to the map and test their effectiveness. Finally we propose some extensions to this work and suggest research in areas not completely covered by our discussion.

1. Introduction

1.1 The Application of Mobile Robots to Mapmaking

Constructing a system of intelligent robotic mapping agents that can function in an unstructured and unknown environment is a challenging task. With the exploration of our solar system as well as our own planet requiring more robust mapping agents, and with the drastic drop in the price of technology versus the gains in performance, robotic mapping is becoming an focus of research like never before. Efforts are underway to send mobile robots to map bodies within our solar system [Krotkov-95]. While much of the research in robotic map construction has been focused on building maps used by the robotic agents themselves, very little has been done in building maps usable by humans. And yet it is the human that drives the need for mapping solutions.

Our goal is to construct precise maps for human usage by improving the software that performs the mapping task. To approach this problem, we must first detail the foundation of mobile robotic agent mapping. A mobile robot, or mapping agent, is a connected collection of sensors and actuators designed to support the agent's ability to detect and measure features of its environment and to allow the agent to move about and interact with its environment. The sensors a mobile robot can carry vary widely. Contact sensors on the robot indicate collision and protect the agent from serious damage by indicating the path is blocked. Simple sonar sensors produce range data that indicates at what distance within a cone protruding from the sensor an object was detected. Sonar sensors are inexpensive and can cover a large area, but are also very susceptible to interference and reflections. Laser range sensors provide for a much more precise distance reading to an object, but their coverage area is limited to a point and they are more costly. Video cameras provide very rich information and cover a wide area, but there is a significant amount of processing that must be done on the data they produce to extract usable information. Additionally, they too are rather costly. Actuator systems on an agent allow it to interact and move about the environment. For movement, we find wheeled and tracked robots. Robots can move forward and backward and they can rotate. To perform the task of mapping, mobility is of the essence, as a stationary robot would only be able to view the world from one position and not detect objects hidden behind other objects. With the benefit

of mobility comes the burden of self-location, the need for the agent to know precisely where it is in the world relative to where it was previously. The more the agent moves, the more important this ability is.

As a mobile robotic mapping agent moves about its environment, it attempts to complete a map of the environment. To determine completion, it needs to have bounds on which space is to be mapped and it needs to recognize which spaces within the bounded region have been mapped and which are still unexplored. In addition, the sensor information collected along the journey about the unknown environment needs to be stored in some form and finally presented to the user once the mapping process has completed. We will call the end product the map. The map is the fusion of all of the sensor data the robotic mapping agent has collected on its mission. Additional data structures can be utilized to also store the information collected by the sensors, but it is the map, the final product, in which we are most interested. Over the past few decades, many researchers have addressed the problem of mobile robotic map construction, as will be outlined shortly. However, most of these efforts have been restricted to single robot agents or imposed restrictions on the environment in which they could operate. These classical approaches fail primarily because of the limitations on the ground truth approach for wide-area navigation in a single or multiple robot system, as will be discussed shortly. We propose a method of map construction by mobile robot mapping agents which has great flexibility. The method can be applied to as complex an environment as is necessary based on the mission parameters or can be implemented in a simple environment without any loss of functionality. It can be applied in a heterogeneous environment where more than one mapping agent is present and each agent can vary in the sensor packages it carries and the physical configuration it has. The method does not rely on the limitations of classical approaches such as [Sing-93], which inspired this work. We accomplish this by converting our gathered data into elements of what we call a *Map Description Language (MDL)*, which is utilized to construct the map. MDL is not a language in the traditional sense of computer science or linguistics in that it does not have semantics and does not require parsing to understand its meaning, as will be described in a later chapter. We want to construct a map of unknown terrain with an unknown set of heterogeneous mapping agents. We want this map to have two features: 1) it is human readable and 2) it is accurate. Let us first examine the key parameters that will affect our ability to achieve these two goals.

1.2 Modeling the Environment

In constructing a map of the environment, we are modeling the space occupied by the bounds of the environment. We must decide how to represent the information contained in our environment, and how to express the spatial relationships between objects in this environment. A mapping agent will encounter two basic entities: occupied space and free, or unoccupied, space. The true nature of the environment surrounding a robot is much more complex than this and the space may not be quite so simplistically represented. As is discussed in [Brooks-85], obstacles can be below a robot's sensors and yet block its motion, or they can overhang free space but allow the robot to pass unhindered. We will treat our objects conceptually as always being detectable by the sensors and always blocking the robot from passing. If a mapping agent can make a transition into a region, this region is classified as free space. If the agent is physically prevented from entering a region by another object, then this region is classified as occupied space. We must decide on how we will represent this free and occupied space in the environment of our robotic mapping agents.

To construct a map from the sensor data we can consider one of two classifications of approaches: grid-based and graph-based. The primary difference between the two approaches is in how the data structures integrate new sensor information and in the type of information that can be extracted directly from the map. Both types of map are data structure representations of the real environment within which the robotic mapping agent operates. These two classes are not specific implementations of environment models, but broad umbrellas describing a basic nature of how the data about the environment maps to the physical world. The two classes utilize the same sensors and actuators to collect data about the environment and organize that data into maps which can be utilized for higher level functions such as navigation and path planning. The classes differ in the way the data about the environment is stored and in how the map data is analyzed to perform higher level robotic functions. Grid-based approaches come with names like 'occupancy grid' [Elfes-90][Singh-93], 'certainty grid' [Moravec-85][Elfes-87] or 'probability grid' [Borenstein-91][Oriolo-95], and graph-based approaches have names such as 'localization map' [Leonard-90], 'geometric representation' [Ayache-89], however all of the techniques fall into one of these two basic classes [Tsubouchi-96].

The grid-based approach utilizes a multi-dimensional array of cells, a grid, to divide the world into discrete units. Each of the cells, or elements, of the grid describes the state of a particular location in free space. Figure 1 depicts a grid representation of an environment. Given the environment on the left of the figure, which contains three objects, we can subdivide this environment into an array of cells, seen on the right. We overlay this grid onto the environment and set the values of the grid elements based upon the state of the environment that each element represents. For example, free space can be represented by a value of 0, and occupied space can be represented by a value of 1. If we indicate the free space with white elements and the occupied cells with grey elements, we arrive at the grid representation of the environment seen at the right of Figure 1. The outlines of the objects from the environment depicted in the left image are only provided on the right image for reference. Those objects and their related curves do not appear in our grid representation of the environment.

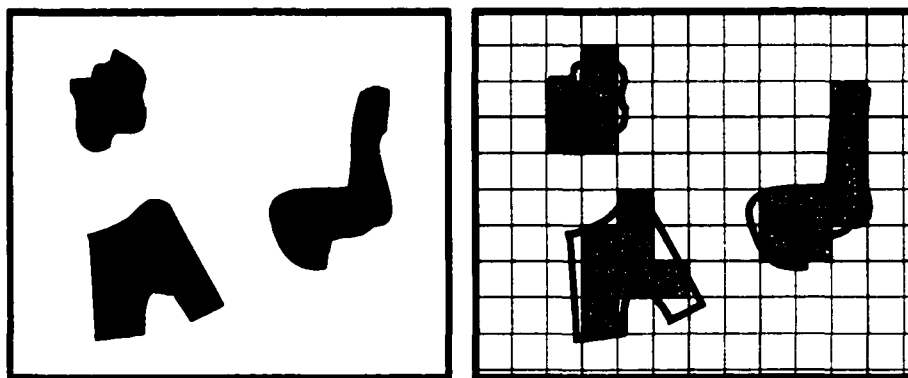


Figure 1. Grid Based Mapping

If we arbitrarily set an origin in the bottom left corner and call that location (0,0) then we can mathematically describe the occupancy of any space covered by our map as follows:

$$Occupancy(x, y) = \begin{cases} Occupied; MAP(x, y) = 1 \\ Empty; MAP(x, y) = 0 \end{cases} \quad 1.$$

where $MAP(x,y)$ is the value of the map grid at index x,y and x increases left to right and y increases bottom to top across the environment.

In our example, we represented the world via a 2-dimensional grid. More complex representations can be accomplished with a 3-dimensional grid, where changes in altitude, mountains,

tunnels and other geographic features not recognizable from view from above, can be depicted. Much work has been done in robotic mapping both for human consumption and for internal robotic consumption. Even if a mobile robot does not want to produce a map as an end product, it frequently must construct one to allow it to move about its environment in an efficient and safe manner. 2-dimensional representations are the most common type encountered due to the lower storage requirements and the simplicity of the analysis of the data in a 2-dimensional grid.

Graph-based approaches take the same spatial information about the environment as grid-based approaches, but store that information differently. Rather than dividing the environment into a set of elementary spaces, the graph-based approach describes the world as a set of nodes and edges. The nodes represent areas of the environment that have a common occupancy characteristic. The edges indicate connectivity between such regions. There is no fixed formula for defining the regions contained within a single node of a graph-based representation. A node describing a region of unoccupied space can be split into two nodes where each new node describes a separate part of the region previously defined by the single node. Figure 2 depicts the same environment utilized from the grid-based scenario, but as a collection of spaces. The left portion of the figure contains the same three objects but shows the free space as being subdivided into nine numbered regions. The demarcation of the regions is arbitrary in this example. The edges of the regions may coincide with faces of obstacles or they may indicate a change in altitude or surface material or the location of reference points in the floor. This subdivided version of the environment can then be translated into the graphical representation seen in the right half of Figure 2. The edges in the graph depict the connectivity between the numbered free spaces in the environment. One can see that from region 3, for example, one can move into regions 1, 2, 4, or 7 but one cannot move directly to region 6 unless one passes through some other regions first.

With the graph based representation, we work with connectivity rather than space occupancy. We can describe the ability to move directly from region S into region D by the following:

$$\begin{aligned} \text{Move from } S \text{ to } D = & \text{ YES; if there exists an edge from } S \text{ to } D & 2. \\ & \text{ NO; otherwise} \end{aligned}$$

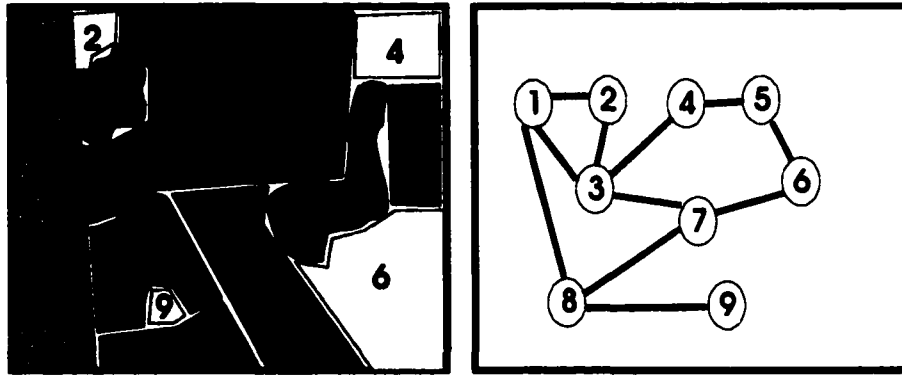


Figure 2. Graph Based Mapping

The conversion of the sensor data into regions is a more complex task than that of dividing the sensor data into a grid of cells. More processing needs to be done at a very early stage to convert sensor information into the type of data that can be stored in the graph-based representation depicted on the right of Figure 2. For example, edge detection is needed to find edges that are then linked to create region outlines. From these regions we then define connected areas of similar properties such as connected free space. With a grid representation, we must only determine occupancy of a grid element and store the value in an array. However, there are significant benefits to utilizing a graph-based map. As graph theory has been a field of research and application for some time, there is a wide array of solutions to problems related to graphs. Navigation is simplified greatly if one can utilize a graph-based representation of the environment where one only needs to find a path along the graph to get from the current location to the desired destination. In a grid-based approach, more significant computational power would be needed to locate and implement a route to a desired destination. However, a graph-based map, while convenient for robot navigation and path planning, is inherently difficult for humans to read and comprehend whereas a grid-based map more directly relates to the way spatial data is normally presented to humans.

Our goal is to provide for a method of constructing a more precise map utilizing mobile robots in unknown terrain. It is therefore important to discuss what affects the precision of the maps produced. As already stated, a mobile robot mapping agent is a collection of sensors, actuators and other physical parts. The sensors gather data about the environment and the mapping agents translate that information into a map. In the case of a stationary robot, all data is referenced to the center of the robot, for example, and as

more sensor data is gathered, it is easy to reference the various data on the map because the geometric relationship of the robot and the sensors is known. With a mobile robot, we also know the geometric relationship between the various sensors and actuators; however, once the mapping agent moves, there is a new relationship between the sensors now and where they were before. Data collected with the sensors after the robot has moved must be treated differently from the data collected before the move. Figure 3 illustrates this very well. As the robot moves a distance D from a starting location, the angle between the center of the sensor and the far-left corner of the obstacle has changed.

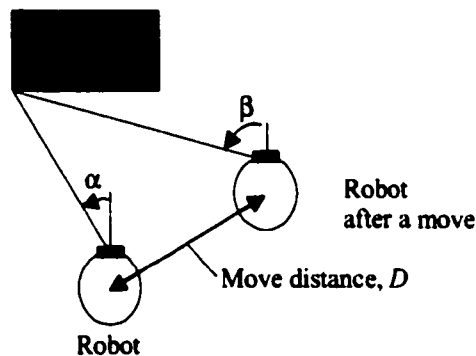


Figure 3. Effect of Robot Movement

When the obstacle appears in the sensor of the mapping agent the second time, there is a change in the location of the entire object as seen from the point of view of the sensor; the angle to the corner has changed from α to β . Likewise, the distance from the object to the sensor has been reduced as the agent moved closer to the object. However the object has not moved, the mapping agent has. The sensor data collected by the robot must be integrated into the map, taking this into account. The mapping agent's position in the environment is utilized for this purpose. The position of the robotic mapping agent provides a frame of reference as the agent moves about the environment. If the position information maintained by the mapping agent were perfectly accurate, then the accuracy of the map would only be affected by the precision of the sensors and the mapping algorithm. The many mechanical and electrical parts interacting to move the mapping agent around its environment combine to cause a drift between the perceived position of the agent in the environment and its actual position. This drift comes from the tolerances in the electrical and mechanical components that make up the robot's drive train, for example, as well as wheel slippage and other traction related problems as the robot moves around its environment. The

mapping algorithm may have instructed the drive train to move the robot forward 10.5 cm, however the robot actually moved 10.4 cm forward. This would result in the robot's perceived position information being incorrect after the move and as a result, any sensor data that is collected at that point would be referenced incorrectly relative to the previously collected data. The approach of keeping track of one's position based on knowledge about how far one has moved is called dead reckoning. The use of dead-reckoning as the primary method for determining the mapping agent's position is dictated by the fact that the mapping agent's environment is unknown and as such there are no reference points or registration markers pre-installed to assist in determining position. The utilization of markers and aides may be useful in a laboratory environment, however they cannot be present when mapping in unknown terrain. As the robot moves around the environment, error creeps into the agent's position information. As this error accumulates, the precision of the map deteriorates. We will introduce a method of storing the sensor data that is collected as the agent moves around that will allow us to easily correct for this error in position once it has been detected and measured. This will allow us to make corrections to the map data collected in the past, so called *historical data*, in such a way that the new map more accurately represents the true environment.

Before we go further into the details of error, let us examine some implementations of the two classes of environment models as they apply to mobile robotic mapping. Table 1 briefly lists some of the recent research in the area of robotic mapping along with key characteristics of the approaches utilized by the respective authors. Details about the way the works relate to our research will be provided in a later chapter.

1.2.1 Grid-Based Mapping

Using methods in the grid-based class, the environment space surrounding the robot is tessellated into a multi-dimensional grid of cells of regular size and shape. This spatial lattice of elements represents the occupancy state of the small parts of the environment that each cell represents. The size of the cells is determined, in part, by the requirements of the completed map, such as detail level and feature size, but should be at least no larger than the smallest mapping agent. The gathered sensor information is used to collect knowledge about *occupied space* and *free space* in the environment (Figure 1). The entire area to

Table 1. Brief Overview of Mapping Research

Authors	Year	Model	Self Location	Data Correction	Key Attributes
Brooks	1985	graph	dead-reckoning, landmarks	Fading	+good groundwork
Moravec, Elfes	1985	grid	Dead-reckoning, Inertial	Only proposed	Representation model, allude to concurrent programming
Elfes	1987	grid	Dead-reckoning, Inertial	-	more advanced world representation
Ayache, Faugras	1989	graph	landmark recognition	PDF models errors	-lines, planes, points
Elfes	1990	grid	dead-reckoning, inertial	error controlled by blurring	good framework
Borenstein	1991	grid	-	-	Solely for obstacle avoidance
Leonard, et al.	1992	graph	Landmark recognition	-	-no self-location
Weigl et al.	1993	grid	-	error controlled by fading	grid: sensors geometric: planning
Singh, Fujimura	1993	grid	-	-	cooperative multi-robotic system
Oriolo, Vendittelli	1995	grid	Dead-reckoning, odometry	-	- dependent on a priori sensor parameters
Santos, et al.	1996	grid	-	sensor specific neural net	fast and robust
Shatkay, Kaebbling	1997	graph	-	-	probabilistic model for efficient path planning
Thrun, et al.	1998	grid & graph	Dead-reckoning, Landmark recognition	probabilistic matching of position and map estimates	probabilistic model for planning paths excellent navigation results -loaded <i>a priori</i> data
Arleo, et al	1999	graph	Dead-reckoning, Landmarks	-	combined local grid and global graph maps -lacks detail of features +variable resolution grids
Davison, Kita	2001	graph	Landmark tracking	probabilistic matching of features	-complexity grows with number of features tracked no treatment of details of features, only their location

be mapped is initially considered *unknown* but frequently assumed as *unoccupied* or *free* and as the mapping agent, in our case a robot, travels through the environment, sensors detect *obstacles* that block the mapping agent. These obstacles are considered solid, occupied space and this knowledge about the occupied and free space is transferred onto the multi-dimensional, tessellated data structure, converting some of the free or unknown space to occupied space and verifying the free or unoccupied status of other cells close by. The grid-based approach attempts to fill the spatial lattice that is the map and model of the environment by attaining complete coverage of the environment such that each cell of the space has been verified as being either empty or occupied through the use of a sensor.

Grid-based solutions to map storage are utilized in work by [Elfes-90] and [Weigl-93]. [Elfes-90] defines an *Occupancy Grid* where each element in the spatial lattice contains an occupancy probability, the probability that the cell is occupied. That work is based on the *Certainty Grid* system of [Moravec-85]. A stochastic sensor model is used; the density for a cell is defined as:

$$p(r|z) \tag{3}$$

where p is the probability that the sensor will report a range measurement of r given that the actual distance to the obstacle is z . Elfes extends this to describe the state of each of the elements in his map grid by:

$$P[s(C_i) = OCC | r] = \begin{cases} 0, & x < r, x \in C_i \\ 1, & x = r, r \in C_i \\ 1/2, & x > r, x \in C_i \end{cases} \tag{4}$$

where $s()$ is a discrete state variable for cells in the grid and can take the values OCC, for occupied, and EMP, for empty. We thus express the probability that cell C_i is occupied, given that the distance to the obstacle is r . This makes intuitive sense as a range reading of r indicates that all cells closer than range r are unoccupied, or have a occupancy probability of 0, while the cell at the precise range reading, r , has an occupancy probability of 1. All cells further away, that is behind, the range indicated by the sensor, have occupancy probability $1/2$, since their occupancy status is unknown. A graphical representation is given in Figure 4 which depicts the probability of occupancy over distances from the sensor to the range reported by the sensor and then beyond the range reported by the sensor.

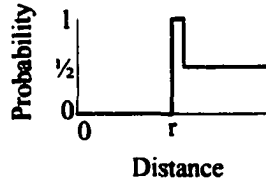


Figure 4. Ideal Occupancy Grid Sensor

A Bayesian estimating procedure is utilized to update the probability data for each cell where the conditional probability can be expanded as:

$$P[A_i | B] = \frac{P[B | A_i]P[A_i]}{\sum_{j=1}^m P[B | A_j]P[A_j]} \quad 5.$$

which results in

$$P[s(C_i) = OCC | r] = \frac{p[r | s(C_i) = OCC]P[s(C_i) = OCC]}{\sum_{s(C_i)} p[r | s(C_i)]P[s(C_i)]} \quad 6.$$

as the formula for a single cell, C_i , occupancy probability. These probabilities are then used to fill the elements of the grid and the robot can utilize their values to make decisions on movement and navigation. The details can be found in [Elfes-90].

Weigl et al. [Weigl-93] utilize a grid-based approach for a part of their solution to the path-planning problem. A grid is employed for the collection and aggregation of sensor data but a graph model is utilized for path-planning. This association of obstacle representation to application is typical in the field of mobile robotics. With [Weigl-93] being concerned with the problem of navigation, a graph model is implemented to assist in the solution of that problem.

Additionally, Weigl et al. addressed a symbolic-based representation, which represents higher level reasoning about the world by identifying objects, which is indicated to be cumbersome for path planning. Elfes also discusses an *inference-grid* model for his solution. There is great value of such a *labeled grid* in that it provides the benefits of the grid-based map while allowing for the attachment of valuable symbolic information that can assist in path planning and mapping.

The grid-based class is a more direct representation of the data collected by the robot's sensors than the geometric based class of methods, since the data returned from sensors conveys occupancy and range data for a section of the environment visible to the sensor. This will become more apparent when we discuss the geometric class. While this class of approaches make the sensor data to map conversion computationally simple, due to the lack of any significant high-level processing of the raw data at the initial acquisition stage, the process of navigation is made a bit more complicated by delaying the analysis of the data which is required for navigation until such later time as it is needed. The higher level processing of the map data, which is needed to assist in navigation and path planning by identifying corridors and paths from locations to goals, is a separate computational model that resides at a higher level of data abstraction within the robotic system and as such is addressed separately. As we shall see, approaches in the geometric class, by the definition of the class itself, perform some of the higher level processing on the sensor data immediately, allowing for a more direct application of path planning and navigation algorithms, as we shall see below.

One of the primary advantages to the grid-based class of environment modeling over the graph-based class is that it facilitates the use of a heterogeneous sensors on a robot, all of whose raw data can more easily be integrated into the common environment database. This is due partly to the very basic nature of the data that is stored in this type of model. When we divide the environment into a grid of zones, each of which is computed to be occupied or vacant, we end up recording a very basic physical characteristic to which most sensor types can relate. Ranging sensors, some of the most common and inexpensive sensor used in mobile robotics and including varieties such as the ultrasonic sonar ranging sensors or laser range finders, obtain distances to the closest object in a given direction. From this we can directly see that all of the grid cells between the sensor and the distance at which an object is detected are empty, whereas the cells at the precise distance measured from the sensor are occupied. Contact and other tactile sensors give us similar information about empty and occupied space. The basic data that we need can be extracted quickly from sensor readings. In contrast, the information needed in a graph-based environment model is not so easily generated from such ranging sensor readings. To incorporate corners,

edges, nodes, and points into a graph model, we need to first extract those abstractions from the raw sensor data, and this requires additional computational work.

1.2.2 Graph-Based Mapping

The second class of environment models is the graph-based class of approaches. The graph-based class can also be described as a *geometric* approach as the information stored within this type of representation captures geometric relationships in the spatial information of the environment. The grid-based class of mapping describes the *unoccupied space* of an environment through the collection of sensor data and exploration by storing geometric representations of the edges of *occupied space*. An alternate format is to geometrically represent the paths or edges of *unoccupied space*, such as was depicted in Figure 2. This is just an inverse representation of the same basic knowledge and transformation from one format to the other is possible; basing the discussion on one of the formats does not affect the results or applicability of those results. The environment is first considered *occupied* or non-navigable in the *unknown state*. Sensor data is then collected which confirms the presence of free, or unoccupied space, and this free space is then modeled geometrically in the map as polygons through sets of edges and vertices. Obstacles or occupied space are also modeled by polygons constructed from edges and vertices and free travel within the map is permitted along these edges allowing unhindered movement by mapping agents from vertex to vertex.

This method is more directly suited for navigation since heuristic and pre-defined models have been employed to convert raw sensor data into geometric models of the world at the initial stages of data collection. This more abstract data can frequently be directly used by path planning and other high-level algorithms, without the need for significant pre-processing. It is thus clear that the problem of navigation is more easily addressed with this class of environment model. The problem of getting from point A to B is simply the problem of finding a path in a graph from vertex A to vertex B along the connected edges. In fact, much of the navigation and mapping using this method is based upon graph theory and as such has a good base of algorithms and research to draw upon since these areas are well understood [Leonard-92, Ayache-89, Brooks-85, Tsubouchi-96].

Leonard, Durrant-Whyte and Cox [Leonard-92] chose to represent the environment of their robot as a collection of geometric features, each of which has attached to it an uncertainty measure and a credibility measure. As the robot moves about its environment, predictions are generated about the geometric features stored in the map and if there is agreement between the predictions and the sensor information gathered by the robot, the credibility measure for that feature is increased. If a new feature is discovered, a new entry is created. However, if predictions about geometric features are not observed by the sensors, the credibility measure of those predicted features is reduced. Thus, as the robot moves about its environment, it encounters three distinct states: matched predictions, unobserved predictions, and unpredicted observations.

One advantage to utilizing the graph-based approach of map storage is the simplicity of applying reasoning to the stored map. By *reasoning*, we mean the interpretation of relationships between elements in the map to be able to form conclusions that give us new information about the environment we are mapping. [Leonard-92] describes a basic system for dealing with dynamic objects in a map by comparing predicted objects with expected objects. Dynamic objects are obstacles that are not stationary. They can occur as sensor glitches, which are not repeated or by other objects moving through the environment. We cannot necessarily determine the difference between a sensor glitch that gives us an erroneous distance reading from a distance reading we take off of an object moving past the sensor. In such cases, we must treat all such readings identically. As a result, we see that the algorithm from [Leonard-92], depicted in Figure 5, can be beneficial for handling dynamic objects in a mapping environment, regardless of the type of map representation that is chosen. We see that unexpected objects are entered into the symbolic map but subsequent lack of observation of those objects, results in a reduction of their credibility until the object is forgotten.

Ayache and Faugeras [Ayache-89] utilize a 3-dimensional environmental model and capture sensory data via passive vision sensors. They utilize vision sensors to convert observed pixel values in the camera data into models of 3D objects such as lines, planes and cylinders. They address a key method that can be used to detect an error in the robot's perceived position and correct that position information by recognizing spatial features detected at different times and locations as being the same physical object. If

we can determine that an object found at location *A* is the same object which is known to be in location *B*, then we can determine the amount of error in the robot's position estimate.

```
for (all predicted and observed measurements)
{
  if observed = predicted
  {
    then increase credibility of predicted target
  }
  else if predicted is not observed
  {
    the decrease credibility of predicted target
  }
  else if observed is not predicted
  {
    then insert new target and begin increasing credibility
  }
}
```

Figure 5. Leonard Map Building Algorithm

Brooks, of MIT's famous AI Lab, proposes that the environment of a mobile robot be represented by relational map which is *rubbery* [Brooks-85]. Brooks proposes some algorithms and also presents only computational problems without supporting algorithms, but the concepts are worth noting. The environment is represented as a collection of *freeways*, which represent unoccupied space that is a zone free of obstacles down which a robot can move collision-free. Additionally, Brooks defines *meadows* as convex regions of free space which do not fit into the straight-line motion areas which freeways describe. Freeways are in fact, merely specific forms of meadows that are of dimension and shape as to be utilized solely for motion between meadows. He describes the notion of an uncertainty manifold, a region within which an object is located relative to some fixed coordinate system. As a robot moves about the environment, the uncertainty about its position, relative to the position it had at the origin, increases and this increases the uncertainty manifold surrounding the robot. It is this area of uncertainty surrounding a

robot which determines its exact location relative to the fixed coordinate system from its origin, which we will call an *uncertainty sphere*. The uncertainty sphere is addressed in a later section in detail.

[Tsubouchi-96] supplies a good survey of mapping techniques in mobile robotics and also classifies mapping approaches into the same two classes, calling them *grid* and *feature drawing*. We direct the reader to this work for a solid overview of the issues surrounding mobile robotic mapping.

The benefits of graph-based approaches are that they are well suited to the application of existing graph solving algorithms and thus graph-based maps are excellent models of the environment for path planning applications where a mobile robot must move from point *A* to point *B* in an efficient and rapid manner. However, the mapping process is a bit more complicated at the early data collection stage as the robot must convert its sensor data into the geometric models which are then combined with the existing geometric map to construct a new geometric map containing the data just collected. This requires basic processing to extract such geometric features such as points, lines, surfaces, and regions at a very early stage, well before a map of the environment is constructed. To accomplish this, additional computing resources and consequently, more time, are required. If mapping for human use is not a mission priority, then the extraction of high-level features can be accomplished more quickly, for example in hardware or parallel software modules and immediately presented to navigation systems and subsequently forgotten. Since long term memory, the basis of mapping, is not needed, and as such, a method for maintaining coherency of data in a map is thus unnecessary, the higher level geometric representations can be discarded once they have served their immediate navigational purposes and recomputed when once again needed. Since our goal is to produce maps usable by human beings and as such, require long term memory on our robotic mapping agents, the use of such basic geometric approaches is not an option.

This does not immediately rule out the use of all graph-based class representations, however, since there are geometric methods that incorporate the higher level geometric features into a global geometric environment model and thus maintain a long term memory. These techniques can be used to produce human readable maps. [Horst-96] provides a mechanism for converting between a certainty grid representation and a object-boundary curve representation of the spatial occupancy of the environment. We can utilize these techniques to convert our grid-based model into a graph-based model for the purpose

of navigation or any other higher level function that may be desired if the system is expanded. However, as our goal is to produce human-readable maps and not to provide for efficient navigation and object recognition, we feel that utilizing a grid-based environment model is most effective for the purposes of this work. Now that we have selected a method of representing our environment and map, let us examine the problems we encounter in trying to build an accurate map.

1.2.3 Combined Grid-Graph-Based Mapping

It should be noted that there is research that attempts to utilize both the grid and graph based approaches, combining them to draw upon their strengths and minimize some of their individual weaknesses. Work by Sebastian Thrun [Thrun-98] developed a system utilizing neural networks to interpret sonar data into occupancy values that were then combined into local and then a global grid-based map of the world. This global grid based map was later interpreted and a graph-based map representation was built to use for navigation. Arleo, Millan, and Floreano [Arleo-99] have also done some work in this area where they generate local grid-based maps and utilize them to construct global graph-based maps that they call topological maps. The purpose is navigation and for this the graph-based approach is ideal, but the graph representation is arrived at via the use of local grid-based maps and the use of variable resolution on those maps to reduce the resource demand associated with grid-based approaches. Their work is somewhat limited in the constraints that all obstacles be flat-sided and aligned with the x-y coordinate axes. The primary motivation for constructing the graph-based maps is the reduced resource requirements of such a world model and the efficient path planning and navigation that can be done with the topological representations.

1.3 Imperfections in Mapping

The primary enemy to any attempt at creating a precise map utilizing mobile robotic mapping agents is uncertainty in collected data. We will also call this uncertainty *error* because the uncertainty in the collected data produces results that are not in agreement with the reality of the environment and this difference between the *real* and the *perceived* is usually described by the term *error*. [Leonard-92] echoes that philosophy by stating that the fundamental problem in robotic map building is that there is error in two elements: the origins of measurements and the values of those measurements. The values of those

measurements are the results returned by sensors and actuators. The origins refer to the registration of the information contained in the measurements to the global map, which must have some agreed upon origin. Leonard et al. suggest error associated with sensor values is well understood and can be handled in the analysis of the data via statistical techniques such as covariance matrices or Kalman filters. The error associated with registration, that is the robot's ability to determine its own position accurately and thus integrate newly collected sensor data correctly into a map, is not so easily treated. [Brooks-85] indicates that all sensors and control systems of a mobile robot have errors associated with them, some of which can be dealt with via calibration of sensors and actuators, however some significant component of error always remains and must be addressed. Brooks refers to an effort by Chatila and Laumond [Chatila-85] that utilized very elaborate techniques to track accumulated error but was still forced to break up detected objects into smaller parts and allow those parts to adjust position relative to one another. This underscores the significance that error control has in mobile robotics and why it is important in our effort of precise map construction by mobile mapping agents.

Irrespective of which type of environment model is used, we will have to deal with error in our map. Consider error to be that quantity of a recorded object which is the difference between what was recorded and what the true condition is that we are attempting to represent. For example, if we use a ranging sensor to compute the distance between two points, the error is the difference between the measurement we recorded from the sensor and the actual physical distance between those two points. Similarly, the error of a map of a building is, in a general sense, the difference between the actual shape of the building in the real world and the shape that is reported on the map we construct. These errors can be expressed using a variety of terms such as imperfections, variances, offsets, deviations, variations or noise. By whichever name you call it, it is a departure from the absolute truth for a particular representation, a reduction in the preciseness of our map. We must address error in order to provide the most accurate map possible. As a consequence of living in the real world, we make no attempt to produce the perfect map and it is understood that we must always endure some degree of error in our maps. It is the degree or magnitude of the error contained in our map that we wish to control. For mobile robots moving around in an

unknown environment and taking readings with various sensors of various features, obtaining precise information about the sensor readings and the robot's position poses a major problem.

As sensors are used and they interact with their environment, their ability to report consistent readings of identical phenomenon can be compromised. Similarly, as a robot moves over longer distances, its ability to determine its position within a global coordinate system, for the purpose of registration of recorded sensor data, also becomes more difficult. This is particularly true if a robot has to proceed through several sequences of movements to reach its current position. It is precisely this kind of error that we will be treating in our research. By allowing for the presence of this type of error and trying to correct for and limit its influence on our final map, we will strive to make our mapping product more accurate and more consistent and reliable in any type of environment, not just restricted to the man made world of laboratory rooms and hallways.

1.3.1 Two Forms of Error

As we try to recognize and limit the effect of error that naturally occurs in both the initial sensor readings and the final map constructed by mobile agents moving in a natural terrain, it is necessary to understand the form and origin of this error. We group sources of error into two categories: *systematic* and *stochastic*. [Brooks-85] refers to them as *systematic* and *random*. We are treating only error in the readings and data reported back by the systems of a mobile robot and not dealing with the mishandling of any data that may be reported, such as echoes from sonar sensors. Some research has been done to handle that form of error: [Leonard-92] uses *credibility measures* to eliminate incorrectly interpreted readings. Stochastic error is the ever-present error that we encounter. It is accepted that it cannot be eliminated [Brooks-85]. It is present because we are using real rather than ideal sensors in a real rather than ideal world. In any real environment with which we interact, there is the presence of this stochastic error, a basic *noise* in any measurements we take and in any interaction we make. We know that there is no way of eliminating it from our sensors, motivators and other components. It is, however, the very nature of the stochastic error that helps us in controlling it. By its very definition, it is random and its effect upon the mapping process can be ignored so long as it does not increase in magnitude to such an extent such that the sensor readings we obtain no longer provide us with data that can be extracted from the stochastic error. If

we have a sensor with a stochastic error of ± 5 units, and the sensor is attempting to read quantities on the order of 1,000 to 1,000,000 units, then the stochastic error is not a serious problem initially. If we are instead trying to measure quantities of 2 to 10 units, then our sensor design is obviously ineffective and the stochastic error renders our readings meaningless. The random nature of this error guarantees that it has no bias (zero mean) and it should thus remain within some bound. If it is discovered that there is a non-zero mean, calibration can be done on the sensor to bring its error to a zero mean.

To illustrate the concept of stochastic error, consider a wheel encoder on a robot's propulsion system that is used to feed back to the navigational system information about how far the robot has moved. The navigational system executes a movement along a path and the encoder returns a distance that the robot has moved. Now, assume that to the value returned by this encoder, we always add a number, K . K is defined by the flip of a fair coin and is equal to -5 if the flip produces a head and a $+5$ if the coin flip produces a tail. The coin's influence is our stochastic error, though simplified. Given that we have a fair coin, the encoder will return a distance traveled that is 5 units too short or 5 units too long for each reading. Over a significant number of readings, these stochastic components will cancel each other out as each of the two variations occurs with identical likelihood. On a statistically long enough journey, the error in distance traveled should be within ± 5 units of the desired distance. The only concern we have is whether the magnitude of K , 5 in this case, is too large with respect to the values we expect to receive as readings from our wheel encoder. If, for expected traveling distances, the wheel encoder returns values in the range of 5,000 to 50,000, then the stochastic error is of little concern to us - its influence on sensor readings is on the order of 0.1%. If the encoder is expected to return values of 3 to 50, however, then we have an obvious problem because we cannot extract any useful information from our sensor readings in the short ranges and only very inaccurate readings at best (10% error). Of course, the significance of any stochastic error that can be treated as acceptable varies from system to system, sensor to sensor and across applications. The requirements of our mapping job, through such parameters as desired minimal feature size of the final map and speed of map construction, for example, will guide us in selecting not only sensor and actuator components for our mobile robots but also what methods are used to utilize them. If very precise mapping of a small location is desired, we can install very accurate sensor and motivator systems

which have a very low inherent stochastic error and also use computational techniques such as measuring features repeatedly and averaging readings, to keep the stochastic error content in our map to an acceptable level. If we cannot afford the time required to take multiple readings and can allow for a greater degree of stochastic error, then we can forgo the multiple readings and still get results in faster time with slightly degraded precision. The Global Positioning System (GPS) is a good practical example of these principles. With a single receiver, a user can drive along a road and get his location information to within some error factor. If that same user stops the vehicle and remains stationary for an amount of time, the readings obtained from the receiver can be averaged and we obtain a more precise position with a reduced error factor. Finally, if differential correction is used, even better results are obtained. In differential correction, a fixed base station of known location to a high precision is utilized in conjunction with the mobile receiver. Since the base station knows its position precisely, it can compute an error component in the signal the GPS receivers are receiving. As this same error is present in the mobile receiver, this correction information can be sent from the base station to the mobile receiver, which can immediately apply it to reduce its position uncertainty on the move.

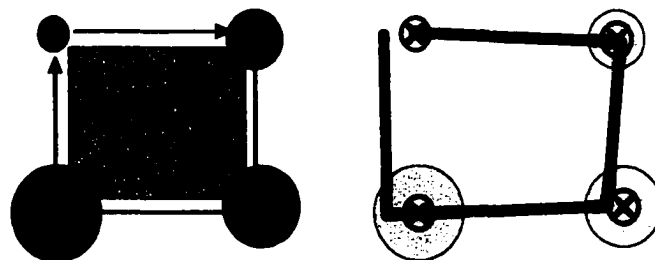


Figure 6. Error Accumulation: Growth of the Uncertainty Sphere

Systematic error is that error component which causes the most trouble to a mobile robotic mapping system and it is this error we will address. Systematic error finds its source in the very nature of the tools and methods we are using to perform our mapping operation. Range sensors, for example, can suffer from temperature coefficient effects that cause a drift in their readings as they heat up with prolonged deployment. Drive systems on a mobile robot traversing unknown terrain will encounter wheel slippage and other traction related issues which result in the robot physically moving a different distance than that which it was instructed to do and also a different distance than which its feedback sensors tell it

that it has moved. The magnitude of such systematic errors can be partially controlled by manufacturing and design tolerances of the components and systems on the robot and the computational algorithms employed to activate the sensors and motivators, record the data, and evaluate its meaning. Unlike stochastic error, systematic error can be biased and is not bound in the same way. The impact such systematic error has on the map a mobile robot builds can increase as the mission progresses until its effect grows to such an extent that the resultant map's usability is severely reduced. As stated earlier, the ability of a robot to know its own location in a global coordinate system is a serious problem, particularly if the robot is relying solely on dead-reckoning via wheel encoders to locate its position. With each movement and with greater distance traveled, the precision of the positioning data for such a robot is reduced. The systematic error component of the position data is increasing because of not only the built-in tolerances of the propulsion system and encoder feedback but also because such systematic error is incorporated into the position data with each position update. This cumulative effect of a biased systematic error can be devastating to the mapping proficiency of a mobile robotic agent if it is not addressed.

Figure 7 shows the drift that can occur as a result of the accumulation of error in the agent's position information. Indicated is the difference in the y dimension between the map the agent constructed and the world model to which it is compared. The mapping agent is indicating the bottom of

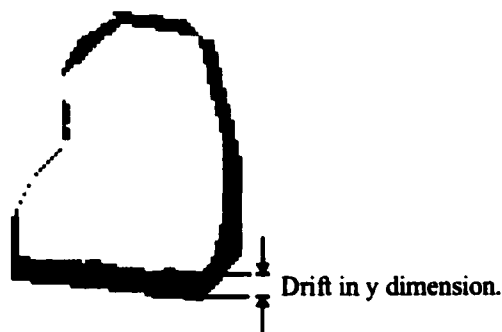


Figure 7. Mapping Drift

the object is much further up on the map than it actually is. The customary representation of error and uncertainty is via a Gaussian error model [Elfes-90, Leonard-90], whereby an error component of mean 0 is added to an ideal sensor reading. The basis for this assumption is the fact that the error that is affecting the data that is being recorded on the map is in fact the combination of a wide range of errors from the

physical tolerances of the mechanical and electrical components and the traction and environmental conditions of the mobile robot. The *Central Limit Theorem* tells us that a collection of such random elements converges to a Gaussian.

1.4 Cooperation in Mapping

Cooperation among robots has become an increasingly interesting research topic in recent time. With most of the research in robotic mapping concentrating on singular mobile robots performing the mapping operations [Borenstein-91][Weigle-93][Elfes-86][Santos-96], it is a worthwhile effort to explore the implications of using a multiple mobile robot system working cooperatively to solve our mapping problem. In fact, a multiple robot system has some inherent features, which lend themselves well to solving some of the very problems faced in mobile robotic mapping. Some basic research has been done in the use of multiple mobile robots to map unknown environments [Singh-93]. Details about the methods used in the above mentioned works will be given in the next chapter.

By the very nature of a collaborative effort of many mobile robot mapping agents, we can address some of the core problems of mobile robotic mapping with new tools. Speed and reliability are key concerns with any autonomous robotic system and as they are for the task of mapping. With many robotic agents working in concert to attack the problem at hand, the terrain can be covered more quickly than would otherwise be possible with a single robot. Reliability is also greatly improved through the redundant nature of a multiple robot system. The failure of any single sensor or mapping agent will not doom the entire mapping mission to failure. The faulty component can simply be removed from the computational system and mapping can continue. Beyond this basic reliability improvement due to the number of robots applied to the task, we have the added advantage of taking into account the heterogeneous nature of the cooperative, multi-robot system. It is by no means required that each robot in the mission be identical. It can be quite advantageous to vary the configuration of the robot agents used as the mapping mission crew. With such a heterogeneous system, not every robot needs the same set of sensors, so a wider array of sensor types can be deployed without expanding the size of the mission. Application or target specific configurations of mobile robots, such as small, low-light sensing robots or larger and faster robots, are of great value in providing for a mission that will result in a more complete map without the danger of terrain

that could not be mapped due to robot configuration conflicts with the environment in which it was operating. If a standard robot encounters an area that is too narrow for it to enter, then it may not be able to explore the region beyond, but a smaller robot can be called in and assigned this exploratory task. Such a heterogeneous multi-robot system could continue on the mission where a single robot or even a homogeneous robot system would fail to acquire mapping information from the region beyond the restrictive opening.

The benefits of a heterogeneous, cooperative, multi-robot system for mapping are clear, however, the complexity of such a system introduces new problems that a mapping technique should address. Typical distributed system problems such as communications, data integrity and data sharing, task allocation and management, and resource management, as they apply to a mobile robotic mapping system, must be considered. This work will not explicitly address these issues in our development or experimentation but the issues should be kept in mind in designing and deploying a real world system.

1.5 Scope of this Dissertation

The goal of this work is to develop a robust, reliable and accurate autonomous robotic mapping system suited to perform in unknown terrain. We want a mapping architecture that will function outside of the laboratory environment while still providing us with a very accurate map of the explored terrain. The mapping product is targeted at human readers and not for robot consumption. We will develop a system which can utilize the inherent advantages of a distributed and heterogeneous set of mobile agents to address some of the major concerns of robotic mapping and make that mapping system robust and produce a more precise result. To accomplish this goal we will need to develop several tools to both simulate the robotic environment for testing and evaluation purposes as well as develop a new method of storing and constructing the robot maps. We also develop a technique for applying correction to our maps through the new storage method. Although we describe a mapping architecture, which will contain many parts, our development and simulation will not utilize every feature described in the system.

1.6 Organization of this Dissertation

In the next chapter, we will review major research that has been done in the area of robotic mapping by others in the recent past and comment on the strengths and weaknesses of those projects in terms of solving the problem we are addressing here: generating accurate, human-readable maps effectively. In the subsequent chapters we will thoroughly define and describe the robot mapping architecture we are proposing and explain how that architecture can take advantage of data storage, distributed systems of robots, and heterogeneous robot groups to construct maps. We will then go into detail on the data-storage method we are utilizing. We will develop an approach for applying correction to the map stored with our new method. Subsequently we will describe a simulator based experimental system used to test the effectiveness of the new data-storage system and also to discuss and analyze the results obtained from that experimentation. We will test a simple and more complex and cooperative mapping mission utilizing the theories developed. We will also discuss the aspects of the robot mapping architecture we are proposing which we have not tested experimentally. Finally we will address the contributions of this work and future research directions which might be taken, based upon this work.

2. Existing Methods of Robotic Mapping

2.1 Methods of Robot Map Construction

As advances in technology have allowed the application and construction of robotic systems that were previously prohibitively large or expensive, the field of robotic mapping has grown significantly. In the recent past many researchers have developed results which have furthered the scope of knowledge in this area significantly [Brooks-85] [Elfes-86] [Elfes-90] [Leonard-92] [Oriolo-95] [Singh-93]. The challenge we face requires the investigation of several problems that have been addressed in various research efforts. Before we present our proposed methodology, let us examine the works of other researchers as they relate to our effort.

In the field of robotic environment modeling, there has been much work, however the aim of acquiring a model of the robot's surrounding environment is not always for the purpose of mapping. Some systems have been developed with the expressed purpose of being able to navigate the surroundings without any interest in producing a map at all. Similarly, mapping methods developed have frequently been designed around a specific type of sensor, or at least been tested with a limited set of sensor types available and tended to focus on solving problems inherent to that sensor technology. In the following section, we will classify each of these methods based upon some criteria which we feel are important. A brief overview is provided in Table 2 and Table 3 later in this chapter, and for the sake of readability, we have divided the set into two groups based upon the environmental model the various approaches employed.

A significant contribution was made by the work of Alberto Elfes and Hans Moravec at Carnegie-Mellon University (CMU) [Moravec-85][Elfes-87][Elfes-86]. They developed the *certainty grid* method of representing an environment based upon the probability that it was occupied and how certain they were of that statement. This approach was later refined into the *occupancy grid* representation by [Elfes-90]. Utilizing the *occupancy grid* model, the environment in which a mobile robot operates is divided into a multi-dimensional field and statistical estimates representing the probability of occupancy and the certainty of that information fill each cell in that spatial lattice. The model was described in Chapter 1 and depicted

in Figure 4. A probability is computed for each cell based on the sensor reading obtained from the robot. Unexplored regions are depicted with an occupancy probability of 0.5, being equally likely occupied and vacant. Correction of error was only proposed in the early work but in [Elfes-90], error was addressed via a blurring technique, which decreases the certainty of mapping from readings based upon deteriorating position information. As the position becomes more and more uncertain, readings are integrated into the map with more uncertainty resulting in any effect they have on the map being less significant than readings taken when the robot's position was known with higher certainty.

Borenstein and Koren [Borenstein-91], at the University of Michigan (UM), did some excellent work in extending an idea developed at Carnegie-Mellon (CMU) by Moravec and Elfes to describe the area that a robot can sense. The sensed environment is described as a *certainty grid* (CG), an array filled with *certainty values* (CV), which represented the probabilistic indication that the area of the real world corresponding to that CV was occupied or not occupied. While the CMU work updated the values of the CG that lie along the arc defined by the reading of a sonar sensor's range result, the UM effort focused on only updating that part of the CG which lies on a line segment immediately in front of the sensor. The certainty grid maps that were generated were utilized directly by obstacle avoidance procedures to assist the robot in navigating around its environment. This approach works well for obstacle avoidance and allows the robot to adjust the level of steering to the probability that an obstacle will be avoided, however the map constructed would not have the high definition of features we would like to produce for human interpretation.

Kenneth Basye wrote a dissertation in 1993 on a map construction framework [Basye-93]. In it, he described three basic aspects of the map construction problem: 1. The Environment, 2. The Agents, and 3. The Tasks. Each of these components require their own investigation and handling to properly solve the mapping problem. In analyzing these three components we come across two trade-offs: timely work and uncertainty. We want to complete the task as quickly as possible, but in increasing the work rate, we inevitably increase the uncertainty in the accuracy of the work we do. Basye approached the mapping problem utilizing a grid-based approach. A polyhedral-based (graph-based) world model was considered, but the problem of implementing uncertainty in a geometric representation as well as the overhead in

integrating new sensor information into such a model made that approach unattractive. With the grid-based mapping technique, one could cheaply acquire data for the map by directly mapping sensor readings without significant data processing. One could also easily implement uncertainty representation and new information could quickly be integrated and old information updated. We agree with Basye's assessment of the strengths provided by the grid-based approach.

M. Weigl [Weigl-93] addressed grid-based mapping utilizing ultra-sonic sensors specifically. They employed grid values as being either occupied or empty but maintained an uncertainty factor related to each cell's validity. The approach is similar to the certainty-grid and occupancy-grid techniques. Each cell's value is governed by the rule

$$h_f + h_o + h_u = 100 \quad 7.$$

where the factors represent the probability of being *free* or *occupied* (h_f or h_o) and the *uncertainty* (h_u) in that interpretation. An uncertainty value of 100 implies that no information is known about the state of the cell; the area has not been explored. As the mapping agent explores the environment and sonar readings are incorporated into the map, a lower uncertainty is given to readings on the axis of the sonar beam and a higher uncertainty to those cells on the edges of the space covered by the sonar beam. This research underscores the strength of the grid-based representation's utilization in mapping. Additionally, Weigl also acknowledges the partitioning of environment representations, though into three groups: symbolic, vector-based, and grid-based. Vector-based corresponds to what we call graph-based. Symbolic is described as referring to objects by their names, their attributes or their topological relations. While it is possible to extract feature knowledge from maps to be able to draw conclusions about attributes and topological relations and then assign those features names, we do not feel that a symbolic representation can exist on its own without either a grid- or graph-based representation underneath. This supports our interpretation of two types of environment representation: graph- and grid-based.

Singh and Fujimura did interesting work at Ohio State University in 1993 [Singh-93] that addressed the issue of cooperative solutions to mapping using a set of more than one robot. They outlined an algorithm for exploring that allowed the assigning of tasks to those mapping agents that would be able to accomplish the task where others would fail. This was based on the idea that all of the robots in the

team were not of the same size and as adjacent obstacles were encountered, it would be possible that larger robots could not fit between these obstacles to explore the regions beyond. The limitations of this work were that it made assumptions on the simplicity of the environment and the total lack of error in collected data or communications between agents. As a result, there are concerns about their algorithm with regard to its ability to operate quickly due to communications requirements. This concern will be addressed in the next chapter when we discuss our approach to cooperative mapping. They also utilized a grid-based representation of the environment.

Oriolo, Vendittelli and Ulivi developed a method similar to occupancy grids in [Oriolo-95]. They also focused on ultra-sonic sensors identifying three primary sources of error from using these sensors. First was error in the measured distance due to the tolerance of the components of the sensor itself. This is the type of error that most frequently comes to mind when imagining error resulting from the use of a sensor. The second form of error was related to the angle of reflection of the sonar beam as the beam will not reflect as well further out from the centerline of the sensor. This leads us to the third form of error – that resulting from false reflections where a sonar echo returns to a sensor after having bounced off of several objects and would lead to an incorrect range conclusion. While we are not dealing with the details of actual sonar sensors and their related problems and solutions in our work, as we are using a simulator for our experimentation, it is still important to recognize the additional work that needs to be done to interpret sensor information before it can be translated into data that is placed on a map. The approach used for computing the probabilities of occupancy and vacancy varied slightly from [Moravec-85]. Oriolo et al. computed probabilities that would slowly ramp up close to the range reading obtained from a sensor and then maintained that high occupancy probability past the range reading, ramping it back down. This gives a wider pallet of probability values rather than the discrete 0, ½ and 1 values generated by the certainty-grid approach [Moravec-85]. Oriolo et al defined their occupancy probability as follows:

$$f_o(\rho, r) = \begin{cases} 0 & 0 \leq \rho < r - \Delta r \\ k_o \left(1 - \left[\frac{r - \rho}{\Delta r} \right]^2 \right) & r - \Delta r \leq \rho < r \\ 0 & \rho \geq r + \Delta r \end{cases} \quad 8.$$

where f_o is the probability that a cell is *occupied* and r is the sensor range reading, ρ is the distance from the sensor, k_o is a constant corresponding to the maximum value attained by the function and Δr is half the width of the area considered *proximal* to the arc of radius r . When compared with the implementation from [Moravec-85] (discussed in Chapter 1):

$$P[s(C_i) = OCC | r] = \begin{cases} 0, & x < r, x \in c_i \\ 1, & x = r, r \in c_i \\ 1/2, & x > r, x \in c_i \end{cases} \quad 9.$$

we can see the subtle difference in how probabilities are assigned, especially past the range indicated by the sensor. While [Moravec-85] treats these regions as unknown, or 50/50 chance of being occupied, Oriolo, et al treat spaces past the sensors reading as probably not being occupied. Both interpretations have merit depending upon how large objects are considered to be.

Santos et al [Santos-96] develop a method of constructing maps local to the mobile robot that are utilized exclusively for navigation within the space and not for constructing maps of the environment. The local maps are computed quickly and as needed and provide a view immediately surrounding the robot that is then used to perform safe local navigation. The idea is to integrate this system into a larger navigation architecture and to have the robot safely move from location to location based on some global scheme. The benefit of the mapping system is it is very fast in acquiring data and constructing a local map but as its sole purpose is navigation, there is no application of the technique to our goal of constructing a global environment map.

Sebastian Thrun developed a combined grid-graph solution that constructed local grid-based maps and built a global grid map of the world from those local maps [Thrun-98]. The global grid map was then used to construct a graph-based version for navigation. Thrun et al [Thrun-98b] developed and later deployed a navigational system that constructed grid-based maps after a human operator escorted the robot by hand through its environment and indicated the location of significant landmarks. The system was deployed in a museum to act as an interactive guide to physical visitors and as a tele-operated guide to internet visitors and performed navigation successfully over 18 km of travel. The *a priori* knowledge requirement and navigational focus does not help our exploring of truly unknown terrain for map

construction, but the successful deployment of the system demonstrates the usability of a grid-based approach in robotic map construction, even for navigation.

With respect to the graph-based representations of the environment, we discovered that the research focused almost exclusively on navigation as the problem being solved. This is understandable as we have already stated that the graph-based model is well suited for solving path related problems and not particularly well suited for integrating sensor data into a maintained long-term map. Leonard, Durrant-Whyte and Cox [Leonard-92] addressed the mapping problem to the extent that it pertains to navigation. They track the location of features of the environment and compare the perceived location of these features with their predicted location based on past observations. If a feature is sensed at its predicted location, then the credibility measure associated with that feature is increased. On the other hand, if a predicted feature is not sensed, its credibility measure is reduced. This technique was described in Chapter 1 and Figure 5. While the environment representation is not suited for sensor data integration or human reading, the results of [Leonard-92] do provide a solution for the problem of addressing dynamic objects in the environment. As objects pass through the environment, they are detected by mapping agents, but should not be registered as static obstacles. Utilizing the credibility updating methods of Leonard et al, we can allow dynamic changes in the environment to take place as the credibility of any moving obstacle will be very low at each location where it is encountered, compared with static obstacles, which will have much higher credibility factors based upon repeated detection at the same location. The solving of the dynamic object problem is critical to any mapping architecture intent on being employed in unknown terrain.

Rodney Brooks of the MIT Artificial Intelligence Lab has done some fundamental work in robotics research and describes issues related to mapping in [Brooks-85]. While Brooks indicates that grid-based approaches are not usable for robots intent on performing many useful tasks in an environment, do to the 2d projection of the 3d world into a plane, he does provide some valuable insight into other facets of robotic mapping. Brooks echoes the notion that robotic mapping is faced with two forms of error: systematic and stochastic and that while some techniques exist to deal with these errors, there are limits to what can be accomplished with calibration for compensating for a detected systematic error, for example. Even with elaborate error tracking systems, mapping approaches of the time resulted in detected objects

being distorted as the mobile robot drifted in its understanding of its own true location. This was related, in part, to the use of an absolute coordinate system. However, in constructing a map utilizing multiple mapping agents cooperatively, it is necessary to have some common frame of reference with which to register the results of each individual mapping agent. An absolute coordinate system is thus unavoidable in the mapping task we are addressing.

Brooks approached the problem of error by providing some formulations to handle the uncertainty of a robot's location in a coordinate system. The idea of a growing sphere of uncertainty as a robot moves through the environment was introduced. A map is constructed out of *meadows* of unoccupied space and *freeways* of unoccupied space that can be used to safely transition between meadows. By growing an uncertainty sphere around a robot's perceived location, it allowed the use of forward reasoning to predict an area in which the robot would end up if known uncertainties accumulated over the path the robot took. Reversing that logic, Brooks reasoned that one could use *backward reasoning* to correct the location the robot is currently at based upon recognizing regions in a constructed map. Working back from that corrected location, it is possible to adjust the previous locations based upon the actions the robot took. This approach of utilizing newly learned information about a mapping agent's location and then adjusting information obtained previously is a core element to our approach to constructing accurate maps.

In addition, several other researchers have utilized the graph-based representation of the environment with mobile robots, but the authors were solving problems other than mapping. Mark Turchan and Andrew Wong did work with acquiring geometric models of objects in the environment for use in solving navigation problems [Turchan-85]. Nicholas Ayache and Olivier Faugeras employed a graph-based model to extract geometric features from sensor data such as points, lines and planes and then utilized these features for landmark recognition, to correct dead-reckoning error, and to navigate an indoor environment [Ayache-89]. Hanna Bulata and Michel Devy also did work on mobile robot navigation utilizing the graph-based representation model. They employed landmark recognition techniques to determine their location in the world [Bulata-96]. Betge-Brezetz et al utilized landmarks detected in their graph-based model of the environment for self-location [Betge-Brezetz-96]. This method provides for the ability to correct the position information in the field and demonstrates the utility of using objects in the

map to detect and correct errors in self-locations. However, the graph-based representation may be ideal for navigation, as it represents all obstacles by ellipsoids, but is not well suited to produce the type of detail-rich map readable by humans that we want to create. Hagit Shatkay and Leslie Kaelbling did work at Brown University [Shatkay-97] on building graph-based maps by tracking the statistical relationships between important points or landmarks. They utilized a probabilistic model relating the robot position and the location of other objects of interest, similar to the work of previous researchers. Specifically they focused on a specialized case of map construction which leads to a very efficient navigational map. Andrew Davison and Nobuyuki Kita did work along similar lines, also utilizing feature correlation and first-order statistics to estimate a robot's position and the position of significant landmarks [Davison-01]. As more features are detected and included, the robot is able to obtain a more accurate estimate of the world state, but more features require more computation as all features are always included in computation even when they are not visible. The details of the map and the details about the features are not treated, as the landmark features and robot are treated as points in space.

Sebastian Thrun approached the map construction problem in a new way by combining both the grid and graph-based approaches depending on which problems are being addressed [Thrun-98]. Thrun utilized a neural network that is trained to interpret sonar data into occupancy grid values and then places those values into a grid-based local map. A grid-based world map is constructed in this fashion. Self-location is addressed by dead-reckoning and correlation with local sensor grid-maps to the current global grid-based map as well as utilizing some a priori knowledge about the nature of the environment (indoor world with perpendicular, flat walls or walls that differed by more than 15 degrees at corners). Once a grid map was built- the map was converted into a graph-based map of the world topology which was used for navigation; the ultimate goal. This combined approach proved quite effective at constructing a navigational map from a grid-based global map in a known environment type.

An extension to this is in the work by Angelo Arleo et al., who extended the combined grid and graph based representations of the world to solve the navigation problem [Arleo-99]. Grid based maps are utilized at the local level and generated from sensor information by use of neural networks. These local grid maps are then used to build and update a global graph-based map directly and this map is used for

navigation in an indoor environment. No global grid-based map was constructed. A novel solution is the use of variable resolution in the grid-based maps being constructed to only provide the level of detail needed to describe the obstacles encountered. With navigation as the goal, details about the objects in the world are lost, insofar as they are not needed for navigation. This does not produce the detailed maps we would like to generate, but it does provide a mechanism for reducing the amount of resources needed to store the grid-based maps. This variable-resolution approach can be considered a form of *map compression* in effect, where otherwise redundant or unneeded information is not stored. The application of the variable resolution principle in practical robots appears to be a good way of reducing the drain on memory resources that can accompany a grid-based model of a large world. Applying high-resolution around obstacles and low resolution in unexplored or open spaces is something to be considered for future use in grid-based solutions in general.

Table 2 and Table 3 summarize the approaches discussed. Key attributes of each effort are described and are indicated as being particularly good (+) or particularly limiting (-). Fields with just a '-' indicate no application of that work to the category.

What can be seen from the tables, in addition to the obvious partitioning of the environment models into two classes, is that the realm of application of the developed methods also falls into some basic groups. The graph-based environment models are rarely used for mapping purposes and best suited for application in navigation and higher symbolic reasoning. This makes sense as we have previously discussed the fact that navigation was a natural companion to a graph based representation of the robot environment. The grid-based approaches find application in the field of mapping. We also see that the most common sensor type used in grid-based approaches was the ultrasonic sonar system. This can partially be attributed to the relatively low cost of that type of sensor along with its well understood characteristics arising from its frequent use. However, graph-based approaches used a wider array of sensors and were the only to employ vision systems. This reinforces the intuitive notion that certain types of sensors are more suited for use with particular environmental models. Sonar and other ranging sensors do well when combined with grid-type approaches due to the nature of the data they return. 3-dimensional

Table 2. Summary of Grid-based Approaches

Authors	Year	Sensor Types	Self Location	Data Correction	Key Attributes	Application
Borenstein, Koren	1991	sonar	-	-	fast, real-time	Obstacle avoidance
Weigl et al.	1993	sonar	-	error controlled by fading	grid: sensors geometric: planning	Mapping
Moravec, Elfes Elfes Elfes	1985 1987 1986	sonar	Dead-Reckoning, Inertial	proposed only	representation model, allude to concurrent programming	Mapping
Oriolo, Vendittelli	1995	sonar	Dead-reckoning, odometry	-	-dependent on a priori sensor parameters	Mapping
Singh, Fujimura	1993	sonar	-	-	cooperative multi-robotic system	Mapping
Elfes	1990	sonar, laser ranging	dead-reckoning, inertial	error controlled by blurring	good framework	Mapping, Navigation
Santos et al.	1996	sonar	-	sensor specific neural net	fast and robust	Navigation
Thrun	1998	sonar, laser	dead-reckoning, a priori environmental information	-	hybrid grid-graph approach +grid based sensor data integration into local maps	Navigation
Thrun et al.	1998	sonar	dead-reckoning, landmark recognition	probabilistic matching of position estimate and map	+very effective for navigation maps -requires human operation to load a-priori data	Navigation

Table 3. Summary of Graph-based Approaches

Authors	Year	Sensor Types	Self Location	Data Correction	Key Attributes	Application
Leonard, et al.	1992	sonar	Landmark recognition	-	-no self-location	Navigation, Mapping
Turchan, Wong	1985	laser ranging	-	-	-assumes geometric models, no dynamics	Navigation
Ayache, Faugras	1989	vision, odometry	landmark recognition	PDF models errors	-lines, planes, points	Indoor Navigation
Brooks	1985	encoders	dead-reckoning, landmarks	fading	+good groundwork	-
Basye	1993	-	-	fast navigation	-	Navigation
Bulata, Devy	1996	Laser ranging	landmarks	Kalman filters	-sensor / environment restrictions	Navigation
Betge-Brezetz et al.	1996	3D laser ranging	landmarks	Kalman filters	+Landmark use	Self Location
Shatkay, Kaelbling	1997	IR, ultrasonic	-	-	constructs good navigation maps	Navigation
Thrun	1998	IR, laser	dead-reckoning, a priori world information	-	hybrid grid-graph solution +grid root for map construction +translation into graph based for navigation	Navigation
Arieo et al.	1999	IR, ultrasonic, tactile	landmarks	-	+combines some grid features and variable resolution	Navigation
Davison, Kita	2001	-	landmarks	Kalman filters	+allows for cooperative mapping -feature tracking adds complexity	Navigation

ranging and vision systems do better with graph-based models as these systems rely heavily on early feature extraction of edges, lines, etc. and such spatial knowledge is more readily gained from those types of sensors.

We do not mean to imply that all mobile robotics research in this area will map in this fashion. We merely state that the segment of published research we sampled displayed this characteristic. In fact, it is likely that concerning oneself with the very environment model used with respect to navigation, mapping or sensor type used will become less and less significant. It is possible to convert environmental models from one type to the other and back again, as needed. Some work in this field has already been done by [Horst-96] and it should be expected that we see further research in this area in the future.

What was discovered was that only some of the techniques in the literature did any work with the problem of self-location and the control of error accumulation in location data and its affect on map construction. Approaches that did not address this concern [Singh-93][Borenstein-91][Santos-96][Turchan-85] assumed perfect location in their approach or did not address the issue. This was usually not a problem as the goal of these methods of robotic mapping was frequently navigation and obstacle avoidance and so no long term memory about occupied and empty space was needed. To achieve the required accuracy in position and sensor readings needed for the short-term memory model for obstacle avoidance and simple navigation filters and other mathematical constructs were frequently applied directly to sensor data or the data fusion stages of the mapping algorithms.

What did strike us was that of those approaches aimed at robotic map construction for long term use (beyond simple navigation), several of the works had little to no provision for correcting errors that accumulated in data collected. While some systems did address the systematic and stochastic error problem superficially, most ignored it altogether. Only two approaches, [Weigl-93] and [Elfes-90], addressed the problem and provided some means to treat it. [Elfes-90] mapped the accumulated uncertainty in position and sensor information to the method used to incorporate new sensor data into the global map. By enlarging, in a fuzzy, probabilistic way, the area in which the sensors detected an obstacle, objects placed into the global map were larger and more vague in their shape (with respect to their occupancy probabilities) than the original sensor data indicated. This was a way of interpreting the uncertainty on

how to reference or register this new sensor data into the global map. [Weigl-93] used a temporal technique called *fading* [Chatila-85][Brooks-85] which reduces the significance of data that had been collected in the distant past relative to newly collected data, unless such readings are re-verified by additional sensor readings. As a result, older readings in the global map begin to fade and blur, and newer sensor readings appear better defined. This approach also provides a good way of addressing dynamic objects in the mapping environment which should not appear as entities on the final map. With the *fading* approach, such objects, as they are only observed in a particular position once or very infrequently, will begin to slowly fade into a *probabilistic mist* over time.

We plan to attack the problem of error in a whole new way. First we ignore it until we can verify its effect upon our map. Then we attempt to correct for its presence to counter the effect. However, we never lose sight of the original map data and as such will be able to revise our *corrections* at a later time if we discover, for example, that the assumed error source was not present or the error was incorrectly attributed. To accomplish this task, we will introduce a new *meta-language*, which we will use to store the data collected by the various sensors on the mapping robots. This meta-language will then be used to construct the local maps for each individual robot in the distributed robotic system. Through the use of the meta-language, we will be able to reconstruct the local map of a robot as it was at any given moment in that mapping agent's journey. This will allow us more flexibility with respect to detecting and correcting for error in both the sensor and position information the robot uses.

With these newly constructed local maps, each robot in our mapping architecture can share what it knows and this shared knowledge base will be used to generate a *global map*. These *global maps* will be made available to the individual robots to assist, not only in navigation but also in process control and error detection and control. In the next chapter, we will describe our mapping architecture in detail.

2.2 Error Detection in Agent Self-Location

A primary component of a mapping solution that performs error correction is error detection, specifically in the drifting position data. The first branch of our solution to the precision mapping problem is to detect the presence of the combined systematic and stochastic error in our sensor data and constructed map. We categorize the methods that can be used to accomplish this task ranging from very specific in

application to general in application. The more specific approaches will be able to make more precise corrections to the data than the more general methods by taking advantage of known information. However, the availability of this known information is not guaranteed so that the more general approaches are more likely to find application in unknown terrain. Table 4 summarizes the three classes of methodologies for detecting error in self-location. Why do we focus on self-location? With a mobile robot moving in unknown terrain, dead reckoning is the most likely system of maintaining self-location information. Errors in self-location, and thus in how new sensor data is referenced and placed into the map, is the major contributor to an erroneous map. Having the ability to detect error in a robot's perceived location is thus crucial to constructing a precise map.

Each of these methods relies on the robot to make a determination at some point, based either on map data it has collected in the past or at the present, or on some outside tool, that there is an error in the map data or in its perceived position information. The reasons why the robot is not where it thinks it should be relate to the error introduced from three major sources: positioning error, sensor error, and communications error. As a robot moves around its environment, it takes sensor readings and these readings are communicated to the central processor on the robot. Physical tolerance and wheel traction problems contribute to positioning error. Surface reflectivity and environmental conditions contribute to sensor reading errors as well as communications errors between systems on a single robot as well as communications between separate robots. As already stated, positioning error is of primary significance when it comes to registration of new sensor data into the map being constructed, and it is this type of error we will concentrate on here. Once this error is detected, the degree to which it is present in the map that the robot is building can be reduced. We will assume that the growth of such error or variance is linear over time although this assumption can be relaxed to allow more complex analysis of historical data if more is known about events that may have happened in the past. For example: assume that it is known that the robot crossed a very small patch of gravel where wheel slippage and the translation of motion commands into true physical movement could have been more compromised. A greater degree of correction to mapping data can be performed on the data collected during that time frame than the rest of the past segment when the robot was moving along on pavement. The reason being that it is assumed that

Table 4. Classes of Error Detection

Method	Application	Example
<p>Return to Feature: This method utilizes the robot's ability to recognize that it has returned to a feature it has recently mapped but the position of this feature is not the same as it was the previous time it was here.</p>	<p>General; the robot need not completely describe the objects it has encountered. If a variation in the position of a point visited before is found, this variation can be used to correct all data on the path the robot took since last visiting the feature.</p>	<p>If a robot travels along a large object then cuts across some free space, maps part of it and then returns to a feature of the object seen previously, we can correct all mapping data collected on that mapping <i>journey</i> by the robot.</p>
<p>Geometric Correction: This method takes advantage of basic geometric knowledge that the robot can determine from the environment and use that knowledge to reconcile and correct the collected data.</p>	<p>Specific; the robot must be able to completely circumnavigate the object in order to determine that it is a closed object. Similar to <i>Return to Feature</i> with additional information available.</p>	<p>If a robot circumnavigates a square, it should end up where it started. If this is not the case, corrections of the mapping data as the robot navigated the perimeter of the square is possible.</p>
<p>Precise Location: If the robot has the ability to precisely find out its absolute position on the map, mapping information in the past can be corrected until the variance between past mapping data and assumed true data is within some predefined tolerance.</p>	<p>General; this can always be applied, but to get a precise fix, if such a positioning system (GPS for example) is available, it may be costly to access. This system can be invoked at any time as it uses real position and assumed local position and does not rely on map data.</p>	<p>If a variation in position from true position is determined, all data can be adjusted in the recent history in a linear fashion to a lesser and lesser extent as you go back in time, until the variance is within some predetermined tolerance.</p>

more error would be introduced into the robot's position relative to perceived position while traveling over terrain with poor traction. This is discussed in more detail in the section on Time Dependent Transforms.

To determine where a mobile robot is currently in the world it is mapping, it must have some way to relate its position to objects in the environment. A robot moves along and keeps track of its position based on a known starting point, via dead reckoning or inertial navigation. As the robot moves around, these estimates of position will begin to drift from the true location of the robot as error accumulates in the position estimate. If a mobile mapping agent can detect some indicator that allows it to compute or verify its position, then it would be able to adjust any accumulated error in self-location.

There has been some interesting research done in methods available for robots to determine their position or recognize their location. Some of these approaches are applied to guidance or navigation only, while others are suited for self-location. The common approach is to utilize a visual system to locate a

feature usable in position determination. Kokichi Sugihara utilized a single camera to compute a mobile robot's position [Sugihara-88]. Eric Krotkov extends this work in [Krotkov-89]. The technique employs a camera that is at a fixed and known position on the robot and the robot is supplied with a correct and known map of the environment. By extracting vertical edges from the image and matching those against the known map, the robot can deduce its location within the map by matching the pattern of edges against what would be seen in the map. Such a technique has little use in an unknown environment, but it demonstrates the use of detecting objects within the map to compute one's location. Mansur Kabuka and Alvaro Arenas utilize, not edges in the scene, but a known, standard pattern to determine the robot's position [Kabuka-87]. A predetermined pattern of interesting features, consisting of black vertical lines and a divided black and white circle, is positioned in a known location within a controlled environment. If the robot can view the pattern from its current location, it can extract visual features from that pattern and knowing the relationship of the camera to the floor and the image to the floor, it can compute the position of the robot relative to the pattern. This approach demonstrates the ability of a mobile robot to determine its position by locating known objects but it is unlikely that we would be able to place visually rich objects into an unknown environment with such precision as to allow their use for computing a relative position.

The approaches above indicate that there is great utility in being able to reference oneself to a known marker and compute one's position relative to that marker. The unknown terrain assumption, however, does not allow the use of precisely located markers such as those employed in those methods. What would be of greater utility is to reference oneself relative to markers that a robot can place in the environment after it arrives. Ideally, these markers would be temporary so that there is no lasting effect to the environment by having the robots map the terrain. In this direction, some interesting research has been done. R. Andrew Russell did some interesting work investigating robot guidance along thermal trails placed by other robots [Russell-93]. Mobile robots were used to lay a thermal trail along a floor and then subsequent mobile robots were able to follow along that trail. A problem that arose related to the decaying of the heat signature which caused varying degrees of oscillation in a robot's trajectory as it attempted to follow that trail. The applicability of such a thermal marker to unknown terrain would not be as simple since the thermal properties of the surface or objects is not necessarily known, or may not be effective. For

example, attempting to lay a thermal marker and utilize it in an environment of high ambient temperature may result in the signature being unable to be distinguished from the natural terrain. Another novel approach along the lines of thermal location was the use of olfactory sensing for robot navigation. Reimundo Deveza et al utilized the application of *smell* to guide a mobile robot [Deveza-94]. The application was similar to the work done in [Russell-93], with mobile robots laying a trail and then having other robots follow along the path, but additional application directions were explored. In addition to having a *pathfinder* lay a trail that worker robots could follow along, retracing ones path and repelling markers were explored. The concept of repelling markers is interesting as it can be utilized to track completion of mapping whereby robots lay an odor trail during their journey and thus prevent additional robots from repeating the same mapping. Such an approach could also be used if a robot detects a serious danger and marks the location as a last-effort similar to a skunk's spray, thus warding off other mobile agents from a similar fate. What these methods of marking give us is a guideline for markers used in unknown terrain. We can enumerate the following set of rules for markers we place in an unknown environment.

1. The marker must decay over time if it is not removed. We take out what we bring in, in essence. The unknown terrain must be treated like a national park. The rate of decay of markers must be guided by the time needed to complete the task at hand.
2. The marker must be invisible and non-toxic to the environment and its inhabitants. The mapping operation should not disrupt or harm the environment.
3. The marker must be inexpensive. This is especially true if many markers are to be placed or the markers are to be abandoned to decay.
4. The markers must be easy to apply and remove. If a marker takes a significant amount of time and resources to apply relative to the task it is designed to assist, its utility may be questioned.

Utilizing markers for mapping purposes is of interest. Mobile agents can use those detected markers to reference their position to the last time they encountered that same marker and thus detect error in self-location, for example. The markers themselves may be coded with more information such as a

universal coordinate or other identifying information, which would assist mobile agents further in self-location.

Other efforts at determining a robot's position in the world depended on having known information about the environment, such as a digital elevation map in work by Talluri and Aggarwal [Talluri-92]. Their solution was applicable to an outdoor environment, but does require a map be given beforehand. The robot's position was determined by searching the elevation map for possible robot locations matching information the robot obtained with a camera, altimeter and compass. Margrit Betke and Leonid Gurvits utilized a known map containing landmarks and then had a mobile robot attempt to identify landmarks in its view and locate them on the map to compute its position [Betke-97]. David Braunegg has done some different work, focusing on building a graph-based model of the world and then using stereo vision to locate the robot, but only so far as to identify a region of the world the robot is in, on that graph-based representation, and not a precise location in a global coordinate frame [Braunegg-93]. Sami Atiya and Gregory Hager utilized a pre-built map to allow very rapid position determination by detecting and locating landmarks in real-time with vision based sensors [Atiya-93].

Current systems proposed for true unknown terrain mapping and navigation are still utilizing dead-reckoning as a means of maintaining position information [Krotkov-95]. Our mapping architecture will utilize dead-reckoning as well, since it is well suited for unknown terrain where no other information is available before exploration begins. The solution implemented for experimentation utilizes waypoints or markers for reference points which combine some of the *Return to Feature* and *Precise Location* characteristics. Our waypoints are active beacons that perform a self-location operation after deployment has begun but before mapping has begun.

The technology available dictates the nature of the waypoints that we may use. For spanning wide-open spaces where the mapping agents and waypoints can see, that is transmit and receive, over the top of objects, one can utilize bistatic RADAR systems technology. In bistatic RADAR, the transmitter and receiver are separate. One waypoint has the transmitter (the origin waypoint, pre-designated as coordinate 0,0,0) and other waypoints have receivers and receive the ranging signal transmitted by the origin. The receiving waypoints can determine the direction of the origin waypoint by measuring signal

strength and obtain distance measurement from the transmitted signal from the origin in conjunction with the synchronized clocks the waypoints carry and that were synchronized just before deployment. Modern RADAR systems are capable of measuring distances down to several meters between transmitter and receiver and can have accuracy of sub-centimeter and would be very useful in detection of position drift. In environments where it is impossible to deploy RADAR type RF ranging systems at will due to obstacle or environmental obstruction, waypoints would need to be line of sight to ensure that there is a line of connectivity from any one waypoint to all other waypoints. By this we mean that if we constructed a graph where each waypoint is a node and an edge is inserted between two nodes if the two waypoints can communicate and range each other directly, then we obtain a graph that has no disconnected components. Starting at any node and traveling along the edges, we must be able to visit every node in the graph. Bistatic RADAR technology, laser ranging technology, or other *surveying* type of techniques can be employed in the waypoints to enable waypoints to determine the distance between them.

The reasoning for building the surveying technology into the waypoints rather than putting this technology into the mapping agents, is that we can thus allow the ranging operation to take significantly more time that would be reasonable if the robots themselves had to do the ranging. As the mapping agents are deployed, they proceed to perform a quick preliminary surveying mission of the terrain, avoiding obstacles and deploying their waypoints at some appropriate and hopefully, strategic locations. As these waypoints are deployed, they can begin to obtain bearings on each other and thus compute their position relative to the origin waypoint, which is deployed first. Once this initial deployment is completed, the robots can begin to perform their mapping tasks. If a RADAR type of ranging system could employ x-ray or gamma-ray technology cheaply and in a compact size, then the issue of line-of-sight would likely be eliminated and waypoints could be deployed at will without regard to their location to other waypoints.

3. The Problem and the New Method

3.1 The Problem of Cooperative Mapping Using Agents

Our goal is to develop a computational architecture that will allow the generation of more accurate robot generated maps for human use. To allow for this we will develop this architecture that can take advantage of a heterogeneous set of mobile robots working in a distributed manner. This dissertation will focus specifically on the method used to store sensor and map data within the individual robot, but we will also discuss how that component fits into a larger conceptual architecture of a heterogeneous, distributed system of many robot mapping agents. For the larger, conceptual architecture, we must address two issues. The first issue is designing the cooperative architecture in which the individual robots operate. Some very good work has been done by [Singh-93] on the subject and we will use their system as a basic framework upon which we will build our architecture. We will enhance this system to try and overcome some of the shortcomings that we have found. The second issue is that of obtaining a precise map. We must first detect error in our map and then compensate for the error and limit its effect on our final map. For this we propose a new method for storing and utilizing the map and sensor data along with a method for incorporating this map data into our cooperative and distributed robotic environment.

Figure 8 illustrates the basic breakdown of our problem and the branches that must be addressed and how they relate. We divide our proposed architecture into halves, each solving one of two issues of cooperative map construction. One branch addresses the construction of a precise map. This half deals with obtaining accurate information and maintaining the integrity of that data once it has been collected. The second branch addresses the issues of cooperation among the multiple agents of our mapping architecture. To produce precise maps, our solution needs to detect errors that affect the quality of the map. This will be accomplished by verification of the robot's position and reconciling that with the perceived location the robot maintains. Secondly, our solution must provide a way to correct for the error once it has been detected. Our proposed technique for storing the mapping data collected by the robot will allow us to make these corrections easily and to *tweak* those corrections if needed. Thirdly, we must be able to control the error that is introduced into our map. In a system of multiple robots, cooperating to

construct an accurate map, there is no doubt that we will not be able to detect or correct all of the error. If we can limit the effect error has on the overall map, we can, for example, prevent the degradation of quality that can result if a system or robot fails non-gracefully.

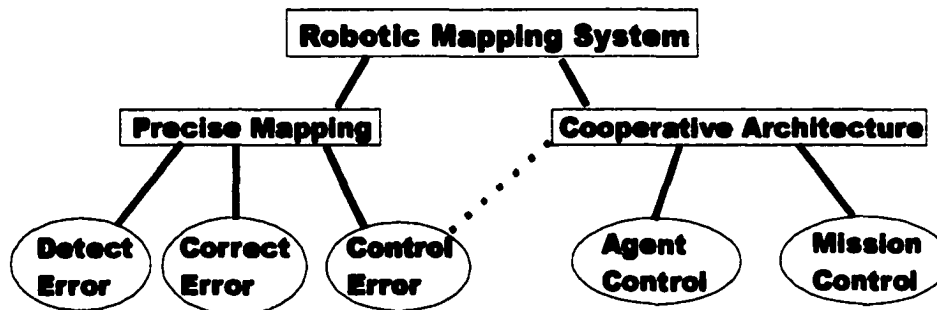


Figure 8. System Overview

The second branch of our system deals with the cooperative aspects of our mapping solution. We are attempting to construct a map through the cooperation of a set of mobile robotic mapping agents, which may vary in their capabilities. To accomplish our goal we must address two major points. First, we must provide for some form of *mission control*, that is, a global algorithm that guarantees that the mapping task will be completed in its entirety and that the task is divided up among the individual mapping agents effectively. Secondly, each mapping agent must provide for a way to share its information with its colleagues and receive similar information from them.

The dotted line between the *Control Error* and the *Cooperative Architecture* segments represents the relationship and contribution of the distributed environment to the task of controlling the error component of the mapping system. This was one of the key reasons for considering a multi-robot system for the mapping task. Beyond the speed and reliability issues, some very important contributions are made via a multi-robot mapping solution. By dividing the mapping task among many mapping agents, we can quarantine error to those parts of the overall architecture where they occurred. This prevents error from one source from poisoning the integrity of the entire map as it is being constructed, something that we could not do if the entire map were constructed with a single mapping agent. This is accomplished by mistrusting the map data that we receive from other mapping agents. We will discuss this topic in greater detail in later sections of this work.

This dissertation treats the branch labeled “Precise Mapping” and the “Detect” and “Correct” leaves with great detail. We explore the “Control” leaf to some extent, however the remainder of the tree is described in the conceptual architecture only, as implementation would go beyond the bounds set for this work. We do implement a form of “Agent Control” and “Mission Control” for the experimental runs but do not investigate the performance of solutions to those problems in this work.

Before we proceed with these tasks, we must lay the groundwork that will be used throughout the subsequent discussions. We will begin by defining some basic concepts and then focus on how we address our two issues: *precise mapping* and *cooperative mapping*.

3.1.1 A Cooperative Agent

Our mapping agents are mobile computing platforms that contain sensors as well as manipulators. The amount and description of the sensor and/or manipulator elements can vary from one robot to the next, as we would like to keep the application of this work as general as is possible. As already discussed, the various systems in a robot contain errors resultant from the physical nature of the components and from the fact that we are operating in an imperfect and complex world. The position information our mapping agents store internally are actually a combination of the true location of the robot agent and the cumulative error up to a given point in time. For example, we express the x-coordinate of the mapping agents perceived location as

$$Pos_x = x + \epsilon_{posx} \quad 10.$$

where Pos_x would be the perceived x-coordinate position of the robot in question and it would be comprised of the true x-coordinate location of the robot in the environment, x , and added to it would be the *error* component, ϵ_{posx} . This error component is the summation of the systematic and stochastic errors in the value of the x-coordinate position. This error has resulted from all of the movement and rotation operations of the robot. Thus, when we use a value such as Pos_x , we are discussing the value that an individual robot has for a variable and not the true physical value that this variable represents. Our experimental results are run on a simulator and this software simulation introduces the error component from predefined distributions. The algorithm implementation portion of the simulator is oblivious to the introduced error in the readings it obtains and the actions it takes and only works with the perceived

values, Pos_x . The error in the perceived position of the robot is of critical importance to us. Since we are mapping unknown terrain with our agents, we have no set of fixed reference points or external sources for position information. As a mapping agent moves around its environment, the error in position begins to accumulate and the perceived position begins to drift away from the ground-truth. Figure 9 shows how the location of an agent is affected by error accumulating in the position information. The agent is the smaller circle with its position indicated by the X at its center. The agent has a radius R and its true position can be anywhere inside of the larger circle such that its extent remains inside that larger circle. For robots that utilize the same propulsion and systems for moving in the x and y coordinate directions, the error components, ϵ_{posx} and ϵ_{posy} , are identical.

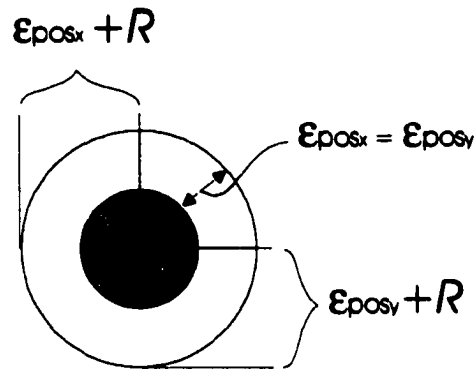


Figure 9. Position and Error Produce an Uncertainty Sphere

The true extent of the robot, physically, is the interior circle. However, due to the accumulated error in position resulting from the utilization of dead-reckoning, the location of the extent of the robot can be anywhere within the larger outside circle. The radius of the robot, as viewed from a fixed and known point in the world, is effectively increased with respect to its possible location on a coordinate system.

Similarly, commands such as movement can also be modeled via :

$$Mov_d = d + \epsilon_{mov} \quad 11.$$

In this case the command for movement of a distance d actually results in a true movement of distance Mov_d . Contributing systematic and stochastic errors such as gearing, wheel ratios and signal delays, as well as wheel slippage from traction problems are all lumped into the constant ϵ_{mov} . The distance reported back by wheel encoders, for example, may not agree with the distance requested in the movement

command. However, neither of those two values may agree with the actual distance moved in the real environment. The robot would register a movement of distance d even though the actual movement in the real environment is Mov_d . The effect is that the mapping agent will degrade its position on the global coordinate system by utilizing this information. The result is that the mapping agent is not where it believes it is and sensor information gathered by it is incorrectly referenced as it is added to the map the agent is constructing.

The combined error component is a combination of systematic and stochastic error, and the net effect will be simulated in our operating environment. The source of the error itself is not of importance to us; we are concerned with treating the result of error entering our map once we have detected its presence.

3.1.2 Modeling Error in Simulations

As already indicated, the values a robot receives and stores from sensor and actuator usage contain the actual value corresponding to the action or environmental reading taken plus an error component. We have divided this generic term, error, into two classes: stochastic and systematic. We feel the nature of the stochastic error makes it difficult to try and counter at the operational, mapping level in a robotic system. It also has a tendency to bound itself to limits of influence. Likewise, systematic error can be controlled to some extent by techniques, but some effects may not be known prior to deployment such as what effect a particular surface material has to related to traction problems in movement, given that the terrain is unknown. We are concentrating our efforts on detecting and correcting for the combined error in our map making efforts.

The first step to controlling the growth of the error content in our mobile robotic mapping system is to find a way to represent it within the system itself. In order to design a mapping algorithm to handle a realistic journey through unknown terrain, it is critical that the simulation system used to test the algorithm models the real environment as best possible. It should reflect all of the shortcomings and problems of the real environment that the mapping robots might encounter. For this purpose, we model the data collected from sensors, communication systems and positioning systems as containing error information that is superimposed on the actual quantities being measured through sensors. We design these systems to be

parameter driven such that the level and form of the induced error can be controlled by the simulation operator to simulate a specific type of environment. The basic error model will be as follows:

$$\varepsilon = \textit{Magnitude} * \textit{RANDOM}(\delta) \quad 12.$$

where the error component is ε and the maximum magnitude of the error is given by *Magnitude*. This magnitude will be controlled via operator parameters as well as predefined relative scales for various components of the simulated robot. *RANDOM* is a random distribution generating function driven by a seed parameter which returns a pseudo-random number sequence between 0 and 1 matching our distribution envelope. This sequence can be uniform, gaussian, or exponential in nature. In our experimentation, we will utilize a gaussian distribution for *RANDOM* and user definable parameters for the *Magnitude*, which will vary from component to component of the robot. The use of a gaussian distribution to model error is an accepted method in the literature [Elfes-90][Weigl-93].

$$f(y) = \frac{e^{-\frac{(y-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \quad 13.$$

The electrical and mechanical systems of a robot would naturally produce a gaussian error component. The stochastic error component can be treated as uniform. The total error component that affects a particular reading or operation which is the combined effort of many electrical and mechanical components and their usage, with the respective systematic and stochastic error from each of those components included, this an error can best be modeled with a gaussian distribution. This is supported by the Central Limit Theorem. The error models in our simulation will be independently configurable for mean, standard deviation, and magnitude so that the error that affects a rotational operation is independent of the error that affects a movement operation or a sensor reading operation.

3.2 Precision Mapping

The goal is to construct precise maps using mobile robot mapping agents. We improve upon the precision of the map by providing a mechanism to correct mapping data when an error in self-location or sensor performance is detected. We will accomplish this through a new map storage paradigm described in the next chapter. As part of the mapping architecture we are proposing, we must also be able to detect the

presence of error. As already mentioned, there will be an unavoidable error component in the data we use to construct our map. Error in self-location will cause problems in how we integrate new sensor data into the map. Data containing error will be generated despite our attempts to correct for it. This will occur through normal operation or can be exaggerated by unexpected events such as sensor failure. The goal is to limit the effect that the data containing large error components has on the total map. Large is a relative term here. The goal is to correct for detected error and to limit this error's effect on the final map that is being constructed. This task will be accomplished via three separate mechanisms. First we must be able to detect the presence of error in the sensor readings and the map we are constructing. Second we must attempt to correct for the error or eliminate as much of it as is practical. Finally we must control the effects of the remaining error in the map that is constructed.

3.2.1 Error Detection in Self-Location Data

As discussed in the previous chapter, the current systems proposed for true unknown terrain mapping and navigation are still utilizing dead-reckoning as a means of maintaining position information. This solution does not rely on any prior knowledge of the environment to be mapped or about the location of any objects within that environment. We will utilize this method of maintaining our mobile robot positions as well. It will be necessary, therefore, to detect error in our position so that we may compensate for it. Of the three classes of error detection described in Table 4 of the previous chapter, the method employed by us will be a *Return to Feature* method with some use of *Precise Location*. We will employ beacon markers that we call 'waypoints', which are deposited by the robots and are active devices and preprogrammed with a location. These waypoints will stay fixed once deployed but can be detected by robots if they come within a fixed range and line of sight of them. This is a form of *Return to Feature* error detection. Since these are our own waypoints, we can make these features as helpful as possible. They are thus active waypoints that can be programmed with a fixed location at which they are dropped and have the ability to send out that information if queried. This additional information borrows from the *Precise Location* method of error detection but is limited as the location information cannot be queried from anywhere on the map and is only practically useful when the robot is very near the waypoint.

As the robot mapping agents are deployed, they fan out into the environment in a pre-mapping mode to deposit these waypoints. A designated waypoint becomes the origin and other waypoints are dropped within line-of-sight of this origin, at which point they can compute their positions relative to the origin and lock their own position, or they are dropped within line-of-sight of a *locked waypoint*. The number of waypoints carried by any single mapping agent is practically limited by physical constraints and so more mapping agents clearly implies more waypoints may be deployed. We will go into more detail on how the waypoints work in the experimental section of this dissertation.

3.2.2 A Hybrid Environment Model

Since our task is map construction and not navigation, and the collection process via sensors lends itself easily to the grid method for data storage, it is the grid based method of map storage that we will use to represent the environment of our mobile mapping agents. However, to assist in the mapping process, we are proposing using a hybrid approach of the grid-based representation, which amends the map with symbolic information, similar to the *inference grid* proposed by [Elfes-89][Elfes-90]. We had developed the notion of a labeled grid-based map independently of Elfes and later discovered that his work also proposed such a solution. This strengthens the notion that such a labeling is indeed of importance. While we do not experimentally test the use of labeled grid-based maps in this work, the utility of such a system is apparent and is part of our proposed general architecture suited for exploring unknown terrain with multiple agents. A mobile robot architecture should utilize a grid-based approach but supplement this grid map with symbolic information, which will be useful to mapping agents in their mapping task.

As discussed previously, it can be of importance to know about the nature of the surface across which one is traveling for the purposes of data correction. Likewise, it can be important to the integrity of a distributed system of robots to know if there are areas of mortal danger to the robots. For example, if the previously unknown terrain does cause a robot failure, the mistake of entering that region should not be repeated by subsequent robots until all have failed.

For this purpose, we propose that the map that is constructed not only be a grid-based model, where areas of occupied space are filled in, but that this map actually be a hybrid between the grid-based approach and a feature labeling system. With this hybrid model, higher level data structures can be

attached to the grid-based map. These *labels* can contain descriptive information about the feature to which they are attached. We will be using a multi-dimensional grid to store mapping information. While RGB data (red, green, blue color) might be stored in this grid for a straight grid-based representation, it is just as feasible to store the occupancy information in one channel and store links to higher-level tags in one of the other channels. This supplemental information will be in the form of tags, which are linked to areas of the grid based map. These tags will contain data on features of the area, which may be useful to mapping agents in determining their position, recognizing features, or navigating (Figure 10). We will attempt to get the functionality of some of the geometric approach through the use of these tags in the hybrid mapping technique. For example, by identifying specific objects such as door or windows, a rapid solution to the problem of "Exit the nearest door" can be found without having to maintain a geometric map of the environment and without the delay of searching the entire grid-map for objects which are then recognized as doors.

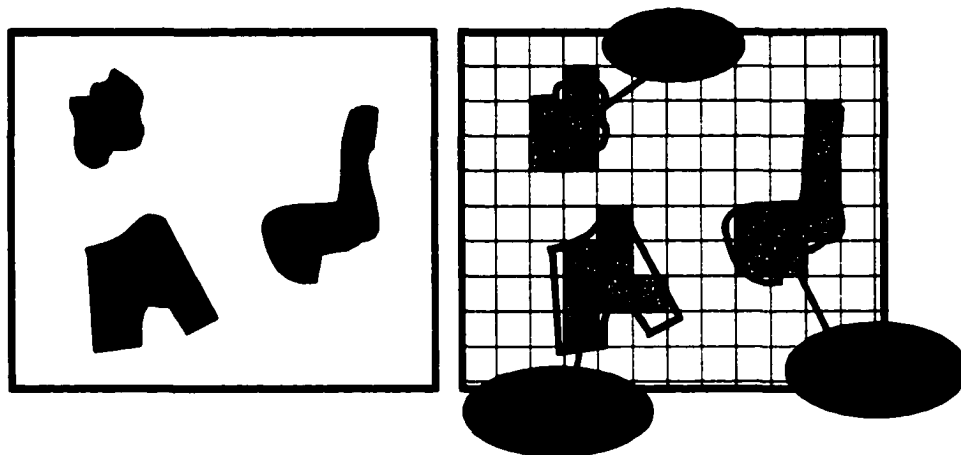


Figure 10. Hybrid Map Representation

Although the grid values in the hybrid-model can be probabilistic such as in the approaches from [Elfes-90] and [Weigl-93], we will utilize binary data in our cells as was done by [Singh-93]. As we will be utilizing a simulator for the robots and sensors, we can control the environment and performance of the components and thus do not need to deal with the complications involved with the characteristics of actual sonar sensors. We can be more confident in the readings obtained from our sensors coming from actual obstacles and not reflections and interference. Using binary cell values will also simplify the analysis of

the performance. These restrictions do not affect the ability to implement and analyze the performance of our map storage and correction solution, nor the architecture outlined.

3.2.3 Error Correction

Once it has been made possible to detect that there is an error in a sensor's readings or the robot's stored position information such that these errors have compromised the accuracy of the map being drawn, it will be necessary to take steps to correct for this error. There are various ways one can make the corrections. Some existing approaches do not bother to correct the data directly at all, but instead evaluate the significance of the collected data differently at various times. The *fading* approach used by [Weigl-93] and the *certainty grid* approach of [Moravec-85] are examples of this sort of treatment. They utilize a *confidence of occupancy* value in each grid element and reduce the confidence of occupancy information filled in where position is known to be suspect, either far from the sensor's focus or sensor readings from a different time frame. We will value all sensor readings as equally significant. It is certainly true that at the time the sensor readings were taken, they were believed to be correct. It may later be discovered that there is a problem with the data delivered, but that should not discount the value of the data as a whole. As such, we will try to correct the data-stream in the past (via historical data) when we detect that there is a problem with the map data that indicates error may be affecting our map. The correction of mapping data collected in the past requires the individual robot to have a historical database of the mapping information it has gathered and used to construct its local map. To facilitate this, we propose a *map description language* (MDL), which is a higher-level data structure than the basic sensor data or grid-based map. This map description language will allow us to convert sensor and positioning information into mapping data *statements*, which describe how the sensor data is to be placed into the grid map. The map description language (MDL) is not a language as traditionally understood in the field of linguistics or computer science. MDL does not have a complex syntax, but a rigid structure utilized for all statements. The term *language* is utilized only because of the sequence of statements that, when *read* together, *paints* a map on a blank canvas. This language can be thought of as analogous to a page description language in computer printers, such as PostScript™. In PostScript, that which comes out on the printer is actually generated from a document of commands that describe objects that appear on the page and how to position those

objects on that page. We think of the data that a robot collects as a stream of *statements*, where each *statement* describes a mapping event. Such events are frequently sensor events where a sensor reading is taken and this data is incorporated into the local map. Along with the sensor reading taken and the time and position at which the reading was taken, a correction parameter is stored. This parameter is initially zero but as an error is detected and correction of the map is performed, these correction factors are set to compensate for the error discovered in the mapping data. Let us examine a conceptual illustration of the concept involved with a map description language statement.

A simple statement might take the following form:

SENSOR, Sonar, Time: 12847, Position: x=1 y=5 z=8, Data=198, Cor=0 14.

Here we know the event is a sensor reading taken from the sonar sensor at a fixed global time of 12847 and position (1,5,8) and the data that was generated by the physical sensors is 198 and there is no correction. The correction is set to zero with the assumption that everything is accurate and functioning with respect to the data we are collecting at this moment. If it were known that this particular sensor had a bias or offset in its result, then we would include that offset at this time in the statement and the correction would not be zero. If this data is later corrected as the result of a discovered discrepancy in positioning of the robot, then the correction parameter would be adjusted accordingly. The correction value is a single numeric value in this illustration, but should be considered to be a more complex object which contains correction entries for all of the recorded values of this map description language statement. The robot uses the complete history of all such map description statements (MDS) to construct or *paint* its local map, whenever called for. It is possible that by picking out specific sections of past statements, only a certain area of the map can be repainted, leaving the remaining map unchanged. This is possible because sensor readings describe not only occupied space but also free space completely and as such both the occupied and empty grid pixels can be regenerated, *covering up* whatever might have been painted in the map, by that sensor, at that location previously. The example is only designed to give a flavor of the map description language. The map description language itself, along with details about the construction of the statements and their components will be handled in detail in the next chapter.

3.2.4 Error Containment

While we attempt to identify and compensate for errors that occur in our map, we do not expect to be able to detect or correct all such errors. To produce a reliably robust system for map construction by mobile agents, we need to address the treatment of error that does remain in the maps constructed by the agents. The cooperative multi-agent solution to map construction provides a useful environment for containing the effect of residual error in our map.

Each robot will store two maps of the environment, a local map, MAP_l , and a global map, MAP_g . The local map will consist solely of the data that a robot has collected itself. This is the map painted from the map description language statements the individual robot has compiled. Periodically, the set of all such local maps will be *fused* to form a global map. Local maps are broadcast from mapping agents and received by all other mapping agents. The collection of such received local maps from the other agents in the field is fused to form the global map, which will be identically stored at each robot. This fusing of the local maps can be defined as follows:

$$\text{Map}_{\text{Global}} = \Gamma \text{Map}_{\text{local}(i)} \forall i, \text{ and } \Gamma \text{ is the fusion operator} \quad 15.$$

The fuse operation is a pixel by pixel operation to generate the *global* map from the *local* maps. We express the fusion in this way to express that the global map stored on each agent is a combination of all of the local maps. Each local map is composed of 3 bands of information. The first and second bands store the *occupancy* value and *known* or *observed* flag values about the environment. The third band stores label pointers to symbolic information, however we did not experimentally investigate mapping utilizing symbolic labels. We label them $b1$, $b2$, and $b3$. Band $b1$ contains the *occupancy value* for the corresponding location with 0 indicating unoccupied space and 255 indicating occupied space. Values between 0 and 255 convey the relative belief in the occupancy of the location; 10 would indicate the space is very likely not occupied and 210 would indicate it is very likely occupied. The local maps contain only values of 0 or 255, based on the sensor results obtained; we have complete confidence in the sensor readings initially. When we combine the local maps, however, areas of disagreement between the local maps can result in cells in the global map containing occupancy values in the range between 0 and 255. Band $b2$ contains the 'known band', which stores the local map information about whether the region of

the terrain has been sensed by the mapping agent. If the $b2$ band value for a location is 1, that region has been sensed and the corresponding region in the $b1$ band has value. If the $b2$ band contains a 0 for that location, then the region has not been sensed and the value in the corresponding $b1$ band is meaningless.

The computation of the elements of the global map proceeds as follows:

$$MAP_{Global}(x, y) = \frac{\sum_{i=1}^n (Map_{Local(i)}(b2, x, y) * Map_{Local(i)}(b1, x, y))}{\sum_{i=1}^n Map_{Local(i)}(b2, x, y)} \quad 16.$$

where $b1$ is the 'occupancy band' of the local map and $b2$ is the 'known band' of the local map.

The effect of the fusion described above is to average those regions which were sensed and explored by more than one agent. This operation was chosen for fusion, as it is the most useful method we could find. In an unknown environment with no other information available other than the sensor data collected by mobile agents, we have no reason to believe any particular agents map contribution is more or less significant or accurate than any other agents. If we only have 2 observations of a region, and they disagree in areas, what basis would there be for giving one agent the benefit of the doubt? Weighing each agent's local maps equally, exempt of other knowledge about the agents, gives each local map an *equal vote* in constructing the global map. If a sophisticated mapping control scheme were developed to detect consistent problems with a particular agents observations, then it would be possible to weight the local maps based on a global *believability factor* for each agent. However, the MDL storage paradigm is designed to allow agents to correct their own data based on detected errors with their sensor readings and so such *believability factors* would simply be contained in the correction factors of the effected agents local map.

A robot continues to collect data and store such data exclusively in its *local* map, using the *global* map only for navigational assistance. The global map is used to guide the exploratory algorithm of the mapping agent to find unexplored regions or to reach them. None of the global map data is ever permitted to be transferred into a local map. By quarantining the global map data from the local map data in this fashion, the data collected by a single robot cannot pollute the data set of any other agents. This is because the only time the global map is consulted by an agent is to assist in path planning and navigation or for

searching for unexplored regions and never for positioning or amending local map data. The only place such erroneous data can end up is in the local map of the robot that generated it, and the common global map. This limits the overall effect that error from a single robot can have on the final map. In essence, each agent does not fully trust the mapping data from any other mapping agent. The *fuse* operation can be thought of as the combining method for the local maps, which is analogous to overlapping and merging all of the local maps. Figure 11 illustrates the flow of map information from individual agents into a local fusing algorithm on each agent which combines these local maps into a global map. This same operation would take place on each mapping agent in the mapping team.

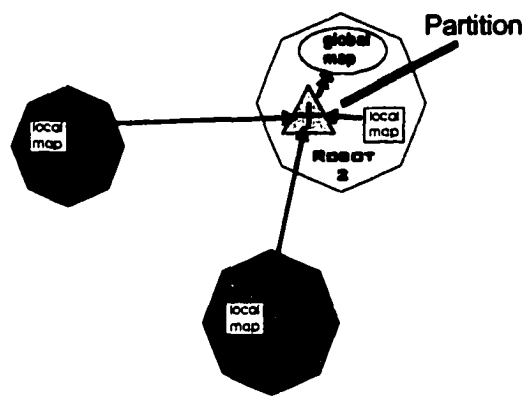


Figure 11. Map Data Merging

The benefits of a multi-agent system can be realized in the merging process. Rather than having the entire map being under the control of a single agent, where any error may well pollute the entire map, we, in essence, split the map into subsets, which are assigned to agents in the team. Each mapping agent is responsible for much smaller segment of the entire mapping task. As local maps are received, agreement in overlapping areas of local maps can improve confidence in the correctness of the data and disagreement can alert to potential faults in a mapping agent. Both conditions can be tagged via the hybrid map representation and thus provide a richer global map to all mapping agents. Most importantly, the merged result is held quarantined from the local map on every mapping agent. This partitioning maintains the integrity of each agent's local map. Erroneous information remains within a single agent and the portion of the global map that agent contributes. Further iterations of global map construction do not cause recycling of the erroneous information via a loop; it can only be resubmitted from the original faulty agent,

which may have detected the error and corrected for it. If not, the erroneous map data is treated as before and only affect that part of the global map visited by the faulty agent.

3.3 Distributed and Cooperative Solutions

As mentioned, there are benefits to dividing a mapping mission among several mapping agents. It allows us to construct a global map on each agent independently. As this global map is the final product of the mapping mission, we are improving the probability that the deliverable can be produced in the light of agent failures. To realize the benefits, we must also address the requirements to maintain and operate a distributed and cooperative system of mapping agents. Good treatments of the communications issues in a distributed robotic environment are given in [Gauthier-87] and [Freedman-85]. We agree with their conclusions and employ a message-passing mechanism for communication among mapping agents, especially considering the very loosely coupled architecture we are proposing. Communications primitives were implemented in the simulation environment, as will be discussed in detail in the chapter on experimental results, and would allow the integration of sensors and processes of any type and from any location, if they complied with the communications protocols defined in those primitives. Henry Fok and Mansur Kabuka have done work on the design of an overall system to coordinate a set of mobile robots in a controlled factory setting [Fok-92]. The principles, that mobile robots need to be capable of performing their own path planning and collision avoidance, which equates to planning their own mapping missions in a mapping problem, hold true. [Fok-92] indicates that the problem of planning motion in an environment of multiple moving robots is NP-hard. Kikuo Fujimura discusses the issues relating to motion planning in an environment with transient objects [Fujimura-94]. Application of similar results to task creation and assignment finds application in our mapping problem. As task assignment and execution need to be completed in real time, only a distributed control system where each mapping agent makes its own task decisions is likely to succeed.

Important work in cooperative, distributed map making was done by Singh and Fujimura [Singh-93]. They have developed a simple architecture to allow multiple robots to cooperate on a mapping task using a grid-based environment model. Their approach will provide us with a framework upon which to

improve. Let us first examine this method. Below is the basic algorithm used for the map making task written in pseudo-code. Each agent in the set is identified by a label, in this case: *robot*.

```
0  Begin
1  MAP(robot)
2  robot.mode=Initialize;
3  While (map not complete) do
4      {
5          IntendedMove(robot);
6          RECEIVE intended-move from other robots;
7          If (ValidMove(robot, intended-move))
8              {
9                  Move robot;
10                 Update partial map;
11                 tlist= CheckForTunnels(robot);
12                 UpdateMode(robot);
13                 For every tunnel on tlist
14                     {
15                         if (!OnQueue(tunnel))
16                             EnQueue(tunnel);
17                     }
18                 SEND map update to other robots;
19             }
20     }
21  End
```

Let us examine the basic logic of this algorithm. Each robot runs through a loop so long as the condition for detecting completion of the map has not been met. The definition of this condition and its implementation are not important to the discussion of this algorithm. Initially, a robot chooses an intended move based on an exploratory algorithm, which each robot has. A simple left to right, top to bottom coverage algorithm could be used, for example. Additionally, the method chosen does not affect the validity of the discussions related to the above algorithm's design. The *intended move* is where the robot wants to go next. Each robot informs every other robot, by means of a broadcast message, about its intended move. As a consequence, after line 6, a robot knows about the intended moves of all of the other robots in the cooperative system. Line 7 performs the conflict resolution for moves that would have two or more robots occupying the same space. If the intended moves and current positions of any two robots conflict, a priority based decision is executed to stall on one or more of the robots to allow the other robot to proceed first. The net effect is that the subsequent movement of the robot in line 9 is potentially

staggered as the conflicts are resolved. After the move has been completed, sensor readings of the robot are integrated into the partial map of the terrain being explored.

Line 11 performs a key component, the check for tunnels. This is important with respect to the heterogeneous nature of our proposed architecture. [Singh-93] define a tunnel as free space between two obstacles where the back end of the free space is undetermined or unexplored. More precisely, it is defined as a 4-connected free space path, with one end proximal to the robot's current location and the other end at the boundary of the robot's sensor-range [Singh-93]. Additionally, the size of the robot prohibits it from exploring the region beyond its sensor range because of the constrictive distance between the two obstacles. The significance and reasoning behind keeping track of these events is that we are using a heterogeneous system of robots, each carrying a potentially different array of instruments and each being potentially different in size, weight, etc. A robot may notice that it cannot continue exploring a particular region due to physical limitations, but it can pass on this information to other robots that may be able to perform the mapping of this region at a later time. The check for tunnels is performed by a robot after it has integrated its new sensor data into its local map. All tunnels found are collected in a list. Subsequently, the robot updates its mode, where it may transition from an initialization mode to a simple exploratory mode (at the start of the mapping process) or from the exploratory mode into a tunnel exploration mode, if, for example, this robot is capable of exploring a tunnel discovered by another robot. This mode affects the move that a robot decides on with the *IntendedMove* statement in line 5.

In lines 13-17, the robot examines the list of tunnels it has collected and checks if this tunnel has already been registered in the global tunnel queue. This tunnel queue is shared by all robots and the contents are distributed by a broadcast to all other robots. If a robot finds that a tunnel it has found is not in the queue, it adds the tunnel to the queue by broadcasting the tunnel information.

Finally, in line 18, the robot broadcasts the updated map information to the other robots so that each robot can update its local map accordingly. All robots thus maintain a map of the environment which is updated not only by their own sensor readings but also by the updated information that they receive from other robots.

3.3.1 Concerns About the Singh-Fujimura Algorithm

The details of how each of the elements in the Singh-Fujimura algorithm are implemented are not important to the discussion of the algorithm unless, for example, one is interested in the complexity of the algorithm. Even so, some important points are worth noting. First, the SEND and RECEIVE operations in lines 6 and 18 obviously come in pairs and are not explicitly listed in the algorithm. Indeed, line 6 should read: SEND intended-move to other robots; RECEIVE intended-move from other robots; and line 18 would read SEND map update to other robots; RECEIVE map update from other robots;. Since all robots are running the same algorithm and not trying to keep secrets from each other, this must indeed be the case. This also exposes one shortcoming of this algorithm: there are two SEND-RECEIVE broadcasts done within each iteration of the basic algorithm. These broadcasts will, in effect, synchronize all of the robots to one another as the appropriate communications sections of the algorithms must be performed at the same time since no robot proceeds further until it has received the intended moves or the updated map information from all of its colleagues. A method of skipping updates from non-responsive colleagues is of importance since a dead or out of contact robot would effectively bring the entire system to a standstill. A time-out or some other mechanism could be used for this purpose however any such system could cause robots to have differing partial maps and this would subsequently affect the ability to determine when the map is completed (line 3 of the algorithm as well as the intended move conflict resolution of line 7). More importantly, [Singh-93] make no mention of how the broadcasts are achieved. In fact, perfect and instantaneous communications conditions are presumed and thus simplify things significantly. We contend that this simplification is not practical and that such broadcast stages must be carefully analyzed for both robustness and speed. Let us assume that the robots were exploring a somewhat large area and a basic collision-detection system of broadcasting the information was used. Such broadcasting works whereby a robot attempts to transmit its information if the airways are clear, but if it detects a collision with another transmission, it stops, waits for some undetermined amount of time and re-attempts transmission. Some communication standards work by this system such as in computer buses and in radio communication. It is conceivable that such an exchange from one robot alone transmitting its information, could take on the order of a second or more (depending upon the number of robots) and that the time for

the entire pool of robots to complete their broadcast will certainly take time on the order of seconds. With such broadcasts happening twice within each single robot move iteration, it is apparent that such broadcasts can substantially slow down the rate at which the mapping operation proceeds. It is thus of great importance to try and limit or make more efficient these broadcast steps in a mapping algorithm, and this is one of the major improvements our proposed architecture contains.

We can eliminate the communications related to the transmission of intended moves and their conflict resolution contained in lines 6 and 7. It is not necessary to check each movement against every other movement if basic collision avoidance is implemented within a mapping agent. Instead, each agent needs to ability to detect a dynamic obstacle, such as a moving robot, or a more intelligent implementation of the movement primitive. If a mapping agent detects an obstacle in front of it while trying to complete a movement operation it can either detect it as a dynamic obstacle, in which case it can slow and wait for the obstacle to clear (give right of way), or if the agent does not have the ability to detect dynamic obstacles, then its movement function can halt for a short random amount of time and then try to complete the requested move. This form of collision avoidance is similar to that used in signaling applications as employed in computer system bus communications standards, networking standards and some radio communications standards. This simple change would result in the following algorithm:

```

0  Begin
1  MAP(robot)
2  Robot.mode=Initialize;
3  While (map not complete) do
4      {
5          IntendedMove(robot);
6          Move robot;
7              (if obstacle is detected in path)
8              {
9                  Delay for random time;
10             }
11         Update partial map;
12         tlist= CheckForTunnels(robot);
13         UpdateMode(robot);
14         For every tunnel on tlist
15             {
16                 if (!OnQueue(tunnel))
17                     EnQueue(tunnel);
18             }
20         SEND map update to other robots;
21     }
```

Additionally, it is clear that the ideal communications conditions assumed in [Singh-93] are not going to be the reality in a natural environment and unknown terrain. It is therefore necessary to expand the algorithm to take into consideration such things as not receiving a transmission from a robot, for any number of reasons, and not accounting for this would bring the system to a halt. The dual-map implementation proposed, utilizing a local and global map which are maintained independently would allow such flexibility. As the agent only utilizes the global map for assistance in navigation and completion detection, there will be no problem if a mapping agent fails to submit its local map to any or every other agent via transmission. This missing information will not be in the merged global map retained on some agents but can be duplicated by the other agents. If this missing information is absent for long enough, the missing area would likely be classified as unexplored and re-assigned to agents for exploration.

3.4 A Proposed Mapping Architecture

We are proposing a system architecture that will facilitate the construction of human-readable maps of an unknown environment by a network of distributed, heterogeneous mobile robots. To this point, we have evaluated several important aspects of such an overall architecture.

We have explored the methods of representing the environment: grid-based and graph-based. It is clear that a grid-based approach offers greater flexibility in the integration of various sensor systems as well as the ability to be more directly understood by human operators. We see that a hybrid representation, which attaches symbolic labels to the grid-based representation, allows us to specify additional information useful not only during the mapping operation but also to the final human client. Data such as terrain characteristics, dangers or environmental conditions or object properties are just some of the symbolic data that may be attached to the grid-based map to provide a more rich result.

We have explored the source and effect of error in a mobile robot on mapping results and how they affect map construction. We focus on self-location via dead-reckoning as a practical and common method of positioning and the drawbacks to it. We see the effect error in a robot's perceived position has on the construction of maps. We have explored the methods of self-location and how they can be utilized to detect errors in position and thus allow for correction of position information. Beacon and landmark

detection was perceived to be the most non-intrusive and practically available aid to self-location in an unknown environment.

We discussed the need to correct the position information in a mobile robot to allow the accurate collection and registration of sensor data to construct a map and mentioned a new method for storing the data generated by a mobile robot which will allow easy correction once errors have been detected. We will focus on the details of this method in the next chapter. We have also developed a technique for containing any residual error generated by individual mapping agents to prevent the contamination of the larger global map. Finally, we discussed a cooperative algorithm for implementing a distributed mobile robot, mapping mission. Let us now present what we believe is a solid framework for a mapping architecture which takes into consideration many more factors not addressed in the approaches presented by past researchers. This system is flexible in its ability to accept new parameters, and it is practical as it reduces the amount of communications between agents in the mapping mission, as compared to the system presented by Singh-Fujimura.

Figure 12 illustrates the framework of our proposed mapping architecture as seen from the point of view of a single agent. The agent receives transmitted local maps from other mapping agents as they arrive and incorporates them into the global map. The global map is used only for obstacle avoidance, navigation and for a completion of mission test. The global map utilizes a hybrid grid map representation to not only store the fused sensor data from the local maps but also the symbolic information pertaining to hazards, environmental conditions and object recognition. No global map information is permitted into the local map. The local map is constructed utilizing the Map Description Language described in the next chapter. This MDL system provides for the correction of any historical sensor and mapping data. The local map also makes use of the hybrid representation and attaches symbolic information to the basic grid based map similar to what is found in the global map.

The mapping algorithm contains several modules. A module to handle the distributed task management is required to divide the mapping task among the mapping agents as well as assign subtasks to other agents in the case of a heterogeneous agent set. We propose utilizing a system built upon the modified Singh-Fujimura algorithm presented in the previous section.

A positioning subsystem keeps track of the agent's location. We are using dead reckoning, as it requires no outside interaction and thus does not impede the performance of the mapping operation.

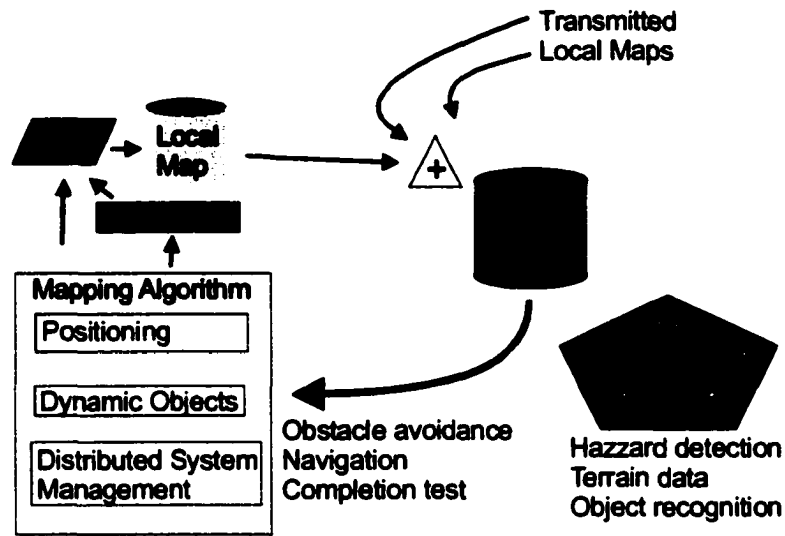


Figure 12. Proposed Mapping Architecture within a Single Mapping Agent.

Occasional correction to the dead reckoning position data via beacons, decaying markers or landmark recognition also allows for adjustment of the position correction factor in the MDL. In our implementation, the beacons are active and intelligent and can communicate their position to the agents that encounter them. The mapping agents do not fix the position information that is encoded in the beacons. Correction of historical map data via the MDL at a later point would adjust the correction factors in the MDL even further. Details on this are found in the MDL discussion in the next chapter.

Finally, a module handles the detection and treatment of dynamic obstacles. Dynamic obstacles are defined as any object in the mapping world that is not a permanent fixture and includes such things as other mapping agents as well as local mobile objects such as animals, mobile machinery, decomposing objects, etc. Handling dynamic objects is significant in the performance of the mapping algorithm already described as it allows for the elimination of the conflict resolution phase of proposed moves in the original Singh-Fujimura algorithm, in that any conflicts in moves can be resolved at the time the conflict occurs. This local conflict resolution takes place between only those agents involved in the conflict and does not hold up the operation of any other mapping agents. We suggest the application of a *believability* index similar to methods proposed by [Elfes-90] and [Weigl-93], which utilize logic outlined by Leonard (Figure

5). Sensors detect objects as present or space as vacant. However if we detect objects as present in locations previously recorded as vacant, we can *paint* them with a brush in MDL, which uses a reduced *believability* to paint in the objects. This reduced *believability* painting results in an increasing of the existing *believability* for the vacant space. Subsequent confirmations of the object would thus repeatedly increase the index of the region until a maximum equivalent of *occupied*. Similarly, the lack of an object where there was a previous detection would reduce the *believability* index of the space and could reduce it down to a lower limit of *vacant*. The local map combining methodology defined in Equation 14 supports this type of treatment of dynamic objects in the environment. The *occupancy* value of a grid location in the local map is always 0 for unoccupied or 255 for occupied. When all such local maps are combined to form a global map, contradictions between local maps over the occupancy of a grid location result in the averaging of the occupancy. If a single agent observed an obstacle at a fixed location which all other agents mapped as an empty region, then the resulting occupancy value within the global map would be much closer to 0, or empty.

If dynamic objects are not explicitly recorded and tracked outside of their effect on the local maps, then the *believability index* mechanism is essentially the same as the already utilized *occupancy value* system for storing the state of each cell. The utility of a *believability index* becomes apparent when dynamic objects are explicitly recognized and possibly tracked and their positions predicted by the mapping agents. This level of dynamic object handling was not implemented in our experimentation.

The mapping architecture outlined above was constructed from the combined knowledge of previous research in the areas in question. We believe that the choices made, which define the architecture, provide for a robust and usable framework on which to construct mapping implementations. We will now focus the remainder of this work on the implementation and testing of a subset of the total architecture. This subset describes the paradigm for storing and utilizing the sensor and actuator information generated by a mobile robot mapping agent and the features associated with its use in a distributed, heterogeneous mapping mission. Figure 13 highlights the subset of the architecture proposed in Figure 12 on which we will now focus.

We will not go into detail the applications of hybrid representation to store additional data with map regions such as danger conditions, surface composition, etc. We will concentrate on the storage of

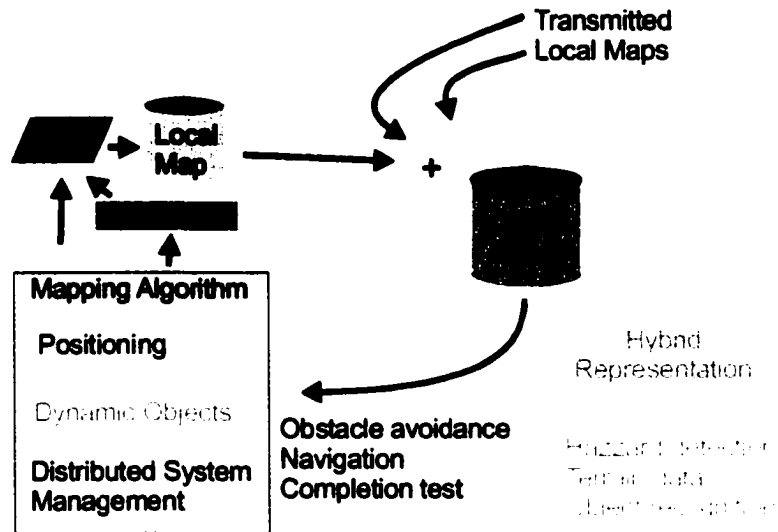


Figure 13. Focus Subset of Architecture

sensor data and robot actions in the MDL format and the use of MDL to paint the local map, as well as position determination and correction and the application of that correction to error correction via MDL. We will then address the multiple mapping agent and cooperative application of MDL.

To implement the MDL paradigm, we devised a simple mapping algorithm to explore the extent of various objects placed in a simulated terrain. Table 5 shows the pseudo-code algorithm used to implement the MDL map storage and correction technique in our simple testing experiments, which implement local map construction and correction with a single robot exploring a simple environment. The map construction component is found in lines 21 through 27, which comprises only a small part of the entire algorithm. Lines 3 through 19 handle the finding of the waypoints for computing the error in the mapping agent's internal dead-reckoning position information, and the use of that error information to set the correction factors of the MDL elements. Lines 28 through 78 deal exclusively with the following of an object's contours in a counter-clockwise fashion by handling convex and concave corners that may be encountered and maintaining a constant distance to the object's side.

Table 5. Single Agent Object Trace Algorithm

```

0   Start
1   While (Not Halt)
2       {
3       if (First Iteration)
4           {
5           Find Registry Point
6           Remember Origin
7           }
8       if (Loop Complete)
9           {
10          Move To Origin
11          Find Registry Point
12          Adjust Correction Factors
13          Halt
14          }
15      if (Found 2nd Waypoint)
16          {
17          Find Registry Point
18          Adjust Correction Factors
19          }
20
21      Sensor(Sonar, Result)
22      PaintSensor(Sonar, Result)
23      Add MDL Entry;
24
25      Sensor(Contact, Result)
26      PaintSensor(Contact, Result)
27      Add MDL Entry;
28
29      if (find object)
30          { /* get parallel to object */
31          Rotate(90);
32          Add MDL Entry;
33
34          Move to Object;
35          Add MDL Entry;
36
37          Rotate Parallel to Object Edge;
38          Add MDL Entry;
39          }
40      else
41          {
42          Move Some Distance;
43          Add MDL Entry;
44
45          Sensor(Sonar);
46          if (Object Ahead)
47              { /* Concave Corner */
48              Back Off Some;

```

(table continued)

```
49             Make MDL Entry;
50
51             Rotate(-90);
52             Make MDL Entry;
53
54             Move Some Distance;
55             Make MDL Entry;
56
57             Find object again on next iteration;
58         }
59     else
60     {
61         if (distance to object < some limit)
62         {
63             Find object again on next iteration;
64         }
65         else
66             { /* Convex Corner */
67             Move Some Distance;
68             Make MDL Entry;
69
70             Rotate(90);
71             Make MDL Entry;
72
73             Move (Distance toward object);
74             Make MDL Entry;
75             Find object again on next iteration;
76         }
77     }
78 }
79 }
80 End
```

The *PaintSensor* instructions of lines 22 and 26 of the algorithm transfer the current sensor readings onto the grid-based map by painting the sensor results with the brush appropriate to the sensor type. Each sensor reading or action generates an additional MDL entry, which is added to the total Map Description List maintained. This list describes the entire contents of the map. As a result, the entire map can be repainted from scratch by issuing the paint instructions for each MDL entry in the list in order from beginning to end. The correction factors adjusted in lines 12 and 18 allow for the correction of the mapping data to compensate for error in the accuracy of the mapping agent's perceived position, which is maintained solely by dead-reckoning otherwise.

To expand this basic system to encompass multiple robots and construct maps of environments with more than one object in them, a different algorithm is utilized. This algorithm utilizes a basic space-filling technique to cover an area of the environment until the agent has reached a dead end or corner and cannot make any more progress with the space-filling routine. At this point, the algorithm searches for some visible but unexplored region within the global map and moves to that location from where the agent begins another space-filling sequence to map this new region. The algorithm is outlined in Table 6. The concept of Time Dependent Transforms (TDT's) is explained in the next chapter in detail. A TDT is a mechanism by which detected error is converted into a correction factor and applied to the elements of the map description, also described in the next chapter. This map description is an form of storing the local map being constructed by each mapping agent.

Table 6. Multi-agent Space-fill Algorithm

```

1  Start;
2  FindRegistryPoint ANGLE and DISTANCE
3  if (RegistryPoint found)
4      { /* found a point */
5      Compute offset of perceived location of RegistryPoint and actual
        location
6      For (each MDL entry in the block we are correcting)
7          { /* adjust each statement's Correction Factor */
8          Compute CorFac based on selected TDT mechanism
9          Set CorFac for MDS
10         }
11         Move block pointer to the end of the MDL description
12     }
13  Sensor(Sonar, Result);
14  PaintSensor(Sonar, Result);
15  Add MDL Entry;
16
17  Result=RobotMove( 4* Size(Robot) );
18  PaintSensor(Move, Result);
19  Add MDL Entry;
20
21  if (Result < 4*Size(Robot) )
22      { /* we could not move required distance */
23      Result=RobotRotate( -90 + (180* DirectionToggle));
24      PaintSensor(Rotate, Result);
25      Add MDL Entry;
26
27      Result=RobotMove( 3* Size(Robot) );
28      if ( Result == 0) Stuck++;

```

(table continued)

```
29             else Stuck=0;
30             PaintSensor(Move, Result);
31             Add MDL Entry;
32
33             Result=RobotRotate( -90 + (180*DirectionToggle));
34             PaintSensor(Rotate, Result);
35             Add MDL Entry;
36
37             Toggle DirectionToggle between 0 and 1;
38             }
39
40     if ( Stuck>1 )
41         { /* deadlock detected - find new region to map */
42         NewRegion = FindRegion(Current Location);
43
44         Result=MoveTo( NewRegion);
45         PaintSensor( MoveTo, Result);
46         AddMdl(&m, fp);
47
48         Stuck=0; /* reset deadlock checking counter */
49         } /* end if */
50
51 Broadcast(Local Map);
52 Build GlobalMap from received LocalMaps;
53 End;
```

Each mapping agent executes the algorithm independently but is synchronized by the reception of the various local maps in lines 52 from which the global map is built. Line 2 looks for any available reference point and if none is found proceeds to take a sonar reading, record the result of the reading and move an additional step in the space-fill sequence. If a registry point is found, the robot communicates with the waypoint and obtains its position from it and the relative position to it. This information is compared with the locally stored global position and any discrepancy is noted and applied via the correction factors to all sensory and action data recorded since the last time a position correction was performed. Details on the correction are given in the next chapter. Lines 21 through 38 handle the reversing operation of the space-fill as the robot snakes its way across the space it is exploring. Lines 40 through 49 are used to detect a dead-end or corner deadlock and then locate a new unexplored region in the global map and move the robot to it.

The algorithms given in Table 5 and Table 6 are not meant to be standard frameworks for implementing a mapping system. They are merely implementations to demonstrate the application of the Map Description Language (MDL) to map construction and how the correction features of MDL are used as well as how multiple mapping agents can cooperate on a mapping mission. For example, the use of specific sensor packages such as sonar sensors, was purely our choice and a variety of sensors could be utilized instead if they perform the same basic operation. Similarly, the choice of moving a specified distance, such as the movement of four times the robots size in line 17 of the algorithm in Table 6, is an arbitrary choice to solve the problem. Other choices would work as well. We will now develop the Map Description Language in detail.

4. Map Description Language – A New Paradigm for Robot Mapping

4.1 Introduction

The Map Description Language, MDL, is the basis for combining data from heterogeneous and distributed sensor and actuator systems in a mobile robotics environment. The goal was to design a method of storing the information gathered from the sensors and actuators on a robot and to allow any robot agent in a distributed and heterogeneous system to be able to utilize that data. Any robot should be able to generate the same map from this data as the robot that gathered the data. Such a system will be inherently more robust as any functioning robot in the system can pick up and continue the work of any other robot in case of a failure; there are no critical systems. A flexible and adaptive system, MDL constantly allows for opportunity and new paradigm inclusion for robot mapping operations. The data sharing among all MDL elements on a platform allows for the exploitation of new approaches to detection and correction problems of previously static data sets through the ability to browse historical information and, in effect, travel back in time.

The basic structure of the MDL is that of a tree. There are a set of syntax definitions that expand the tree from a single node, however, the actual MDL sentences can range for simple to complex depending upon which elements of the MDL syntax are actually utilized to describe the data of a specific event. Each and every element of the MDL sentence can be NULL or empty. This pruning of unneeded parts of the MDL sentence syntax can simplify the MDL forms of some data. Conversely, the tree nature allows complex data representations to be built, which can encompass every imaginable aspect related to a robot operation. Statements in MDL can range from simple to complex (Figure 14).

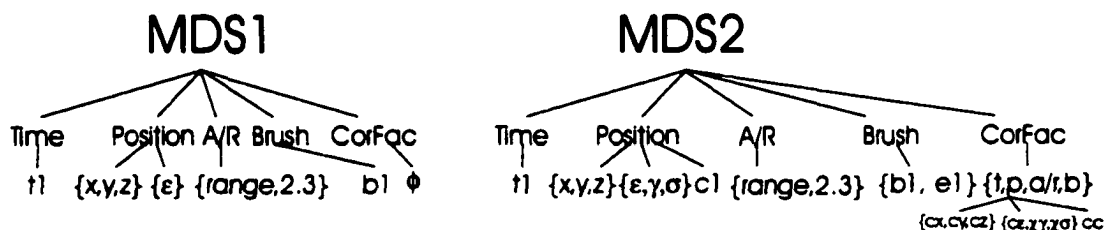


Figure 14. Simple and Complex MDL

An entire tree is required to model a sentence in the MDL, which we will call a Map Description Statement (MDS). All of the data collected by a robot is represented by a sequence of MDS constructs and the map can be *painted* by parsing through the MDS elements. The maps constructed by each robot in a distributed environment can then be combined to represent a total map. This description of the map a robot is building through exploration can be described as

$$Map_Global = \Gamma_{\forall j} Map_Local_{R_j}, \text{ where } j \in \{1 \dots J\}, J = \# \text{ of robots.} \quad 17.$$

$$Map_Local_{R_j} = \Psi_{\forall i} MDS_i \text{ where } i: 0 \dots \# \text{ of MDS. } I \in Time = \{T_A, T_B, \dots\} \quad 18.$$

where Γ represents a logical combining operations that is defined dependent on the type of elements it is combining. For the construction of the global map from local maps shown in Equation 17, the meaning is as defined in Equation 16. The Ψ operator represents the combining of the sequence of MDS elements into a local map, as expressed in Equation 18, and the meaning is the algorithm which interprets the values in each MDS and paints those meanings into the local map.

In our distributed and heterogeneous mobile robot mapping system, each robot agent builds an independent local map using the MDL statements representing its own sensor and actuator data. These statements are time dependent and this is why the collection of the individual MDS elements is not just a set but rather a sequence. The significance of the ordering of the statements will be discussed later. The local maps can then be transmitted to other robots where any and all robots then fuse these local maps into a global map of the entire environment. Each robot agent keeps these global maps separately from the local maps, for reasons that will also be discussed later. The structure of each MDS is a simple 5-tuple:

$$MDS = (Time, Position, A/R, Brush, CorFac) \quad 19.$$

A/R: Action-Reaction Pair, CorFac: Correction Factor

Each MDS is designed to store the information that is represented by a single event in the robot's operation. The most easily recognized event is that of firing a sensor and recording the reading. The MDS will store this reading of the sensor for future use. Other types of MDS include commands to move, which by themselves do not seem to generate any mapping data directly but are important as we will see shortly.

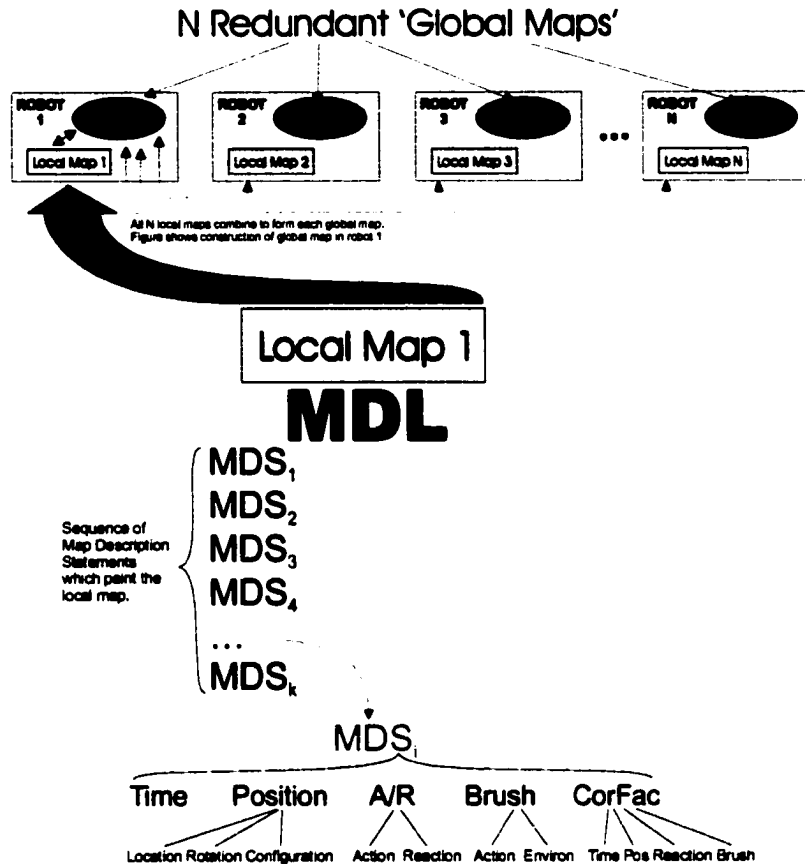


Figure 15. Overview of the MDL Mapping Technique

Before we proceed with a detailed definition of each of the elements of the MDL, let's look at an overview of how the various elements we will be defining fit together. Figure 15 outlines how the various MDL elements and their sub-parts relate to create the local maps on each robot independently. These local maps are then independently combined to form global maps.

4.2 Time Element

Let's examine each of the five elements of the MDS statement syntax sequentially. The *time element* is significant because it serves to sort the MDL statements chronologically. The reasons for keeping the MDL statements in chronological sequence will be discussed later. The time element itself is nothing more significant than a time-stamp. The nature and format of the time-stamp is unimportant and each of the robots in the heterogeneous system need not use the same format or clock; there is no global clock. The contents of the time element are important only to the local robot, which is gathering the MDL

statements to build a local map. The only requirement is that the time elements form a monotone non-decreasing sequence for each robot. The reason for this is to maintain the order in which the events occurred as we will have the ability to apply time varying transforms to the MDS sequence at a later point to perform corrections for anomalies that are discovered. We can define the time element simply as:

$$\text{Time} = T_e \text{ is the time the event or action occurs.} \quad 20.$$

4.3 Position Element

The *position element* of the MDS is used to store the location of the robot. It represents the location of the robot when the data encoded in the MDS was recorded. The nature of the position element is customized to the application. The position element itself is a 3-tuple:

$$\text{Position} = (P_L, P_R, P_C), \quad 21.$$

where P_L describes the location of the robot and is itself a 3-tuple in a coordinate system, P_R describes the robot's rotation in space and P_C describes the robot's configuration from a finite set of predetermined configurations the robot can take. P_L , the location element, encodes the location in three-dimensional space of the robot. If we are using the Cartesian coordinate system, then the 3-tuple contains the x, y, and z coordinates of the robot based upon some universal origin, which must be common among all of the robots in the system. We thus have $P_L = (P_{Lx}, P_{Ly}, P_{Lz})$ in our Cartesian coordinate system, each element representing the x-, y-, and z-coordinates respectively. We are free to simplify this representation (or expand it, if more dimensions are needed) as the situation requires. For simple laboratory experiments, it is frequently sufficient to use a two-dimensional coordinate system. In this case, P_{Lz} is always constant and can be treated as *null* (\emptyset). In our tree representation, the location part of the position element only has two children in such a system rather than three. This illustrates the way the MDL can be configured to suit the requirements of the system.

P_R describes the rotation of the robot about its own axis and affects the interpretation of sensor data as it determines in which direction from the current robot location the sensor is taking its readings. In three dimensional space, a 3-tuple representation of P_R makes sense for storing the rotation of the robot about each of the three major axes, however, for simplicity we will assume only one axis of rotation, about the z-axis, and thus simplify the rotation element to a single value. It is clear that the size of the rotation

element, as with most elements, can be enlarged to capture the greater complexity of a more complex robotic system.

P_C describes the configuration of the robot mapping agent and the value of this element is taken from a set of allowable configuration settings. The necessity to track configurations comes from the possibility of a robot mapper to reconfigure itself and thus alter the relative relationships between the various sensors and actuators on the robot. Such a reconfiguration would obviously change the meaning of any sensor data collected as is illustrated in Figure 16.

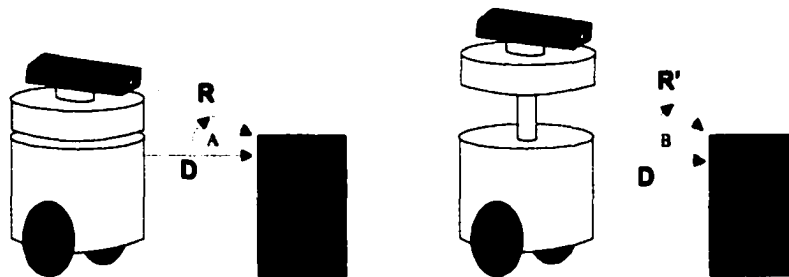


Figure 16. Robot Configurations

The configurations in Figure 16 show the sensor on the top of the robot in two different locations, the right one at a much greater elevation than the left one. If the sensor is aimed parallel to the plane on which the robot rests, then the objects the sensor *sees* can obviously vary and as such the data returned by the sensor in both configurations must be evaluated with this in mind. Both are the same distance, D , from the object, but one robot sees the range from the sensor as R (angle A) and the other as range R' (angle B). Clearly $R \neq R'$ and $A \neq B$. It is possible that both configurations see a specific object but at different altitudes and we may thus obtain contradictory sensor data which may mislead us if we ignore this configuration change. With the term *configuration*, we mean to describe the specific physical settings of the robot at a given time. Changes in configuration may be required for a robot to fit through a physical opening, for example. Readings taken while contracted may not maximize the sensor's capabilities but still return useful information and this may be the only data collected on a region due to its physical characteristics. We must capture the ability of the robot to move its sensors relative to one another so that the information gathered by these sensors is interpreted correctly rather than leading us to assume we are receiving contradictory data from an earlier reading. If all of the robot's sensor and actuator systems are

physically mounted so they cannot move relative to each other, then the robot effectively has a single configuration only and this parameter becomes trivial. The set of configurations a robot can take can be known *a priori* by the robot's construction or can be determined in the field dynamically, thus necessitating storage for only those configurations used. We describe the configuration as coming from a set of specific configurations and we can thus reference the configuration from the set rather than having to include the exact details of the entire configuration as it pertains to all of the robot's systems. This can be described as follows: $P_C \in \{ P_{c1}, P_{c2}, \dots \}$ where each element of the set is a different configuration and the specifics of the configuration needed to properly interpret the data collected can be called up based on the configuration numbers 1, 2, 3 and so on.

The configuration aspect of robot *status* at the time of any sensor reading is significant as there is work going on at research centers at this time which utilize reconfigurable robotic systems. NASA and the Jet Propulsion Laboratory are working with robots in the field that can reconfigure to assist their traversing of rugged terrain (Figure 17) and reconfigurable applications are envisioned for planetary exploration purposes as well (Figure 18).

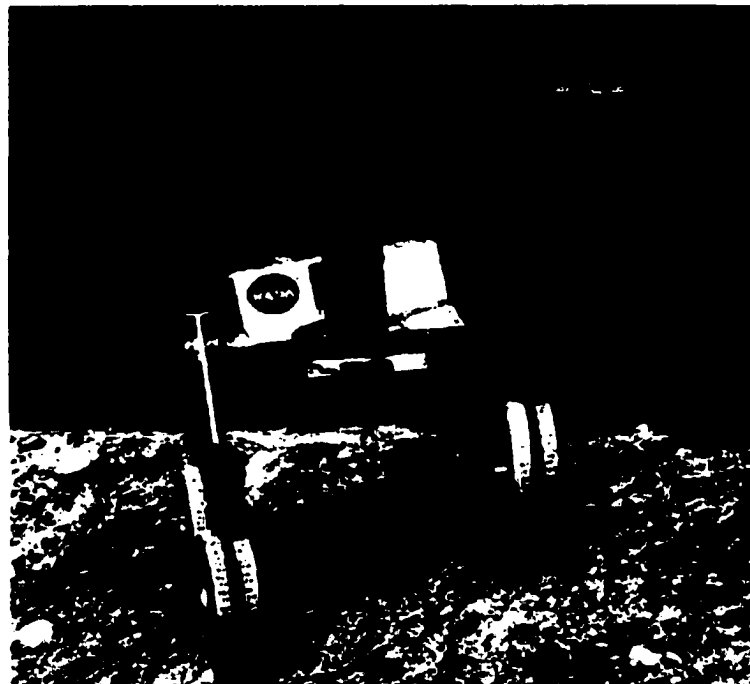


Figure 17. Reconfigurable Robot Developed by JPL, CalTech [Schenker-00][Schenker-01]



Figure 18. Another Small Reconfigurable Robot Developed by JPL, CalTech [Wilcox-96]

This completely describes the *position* element of the MDL statement. In simple laboratory exploration with simple robots, the complexity of the position element can be much reduced. If we are traveling on a single plane such as a building floor, we can simplify our location factor to just x and y coordinates. Similarly, a single rotation around the *vertical* axis of the robot reduces the rotation element to a single term. Finally, if the robot has no movable sensors or actuators, then it has only a single configuration and the configuration element is a constant and can be ignored. This would all result in a position element of the form $P_R = \{P_{R1}\}$, $P_L = \{P_{Lx}, P_{Ly}\}$ and thus Position = $\{ \{P_{Lx}, P_{Ly}\}, \{P_{R1}\}, \emptyset \}$.

4.4 Action/Reaction Element

The action-reaction pair or *A/R* element is the core of the sensory and operation data record stored for each event through the use of the MDL. This is where the actual information returned by sensors or operations by wheels is stored. As the name implies, the *A/R* element is a 2-tuple consisting of an *action* and a corresponding *reaction* based upon that action. The action is that part which the robot mapping agent initiates such as a request to fire a sensor, move a wheel or run an internal diagnostic. The reaction is the resultant response from the action and can take the form of a status or pass/fail indication or a stream of data from a sonar array, for example. If the action was a command to move forward 3 meters, then the reaction can be the data read from the wheel encoders indicating distance traveled. To express this more formally we can say that:

A/R = { Action, Reaction}, 22.
Action \in { robot commands },
Reaction \in { feedback or results from robot commands}.

The following example illustrates this concept. The robot issues the action “fire sonar sensor #4”, the reaction to which is “1.2 meters”. The action “reboot CPU b” receives the reaction “System Ready”. Finally, the action “move +3.5 meters” generates the reaction “moved 3.48 meters”. It is important that we have access to the *intended* as well as the *accomplished* action because we will be able to use this data to not only *detect* errors in the map being constructed, we will also be able to *compensate* for these errors through a sequence of time-dependent transforms on MDS entries for the map.

4.5 Brush Element

The brush is tightly related to the A/R element in that it determines *how* the information stored in the A/R element is transformed into data that is integrated into the local maps. The brush defines how to *paint* the reaction data from the A/R element onto the local map. The brush interprets the method of painting the data into the map based on two critical elements: 1) the action and 2) the environment. The action determines the type of sensor and consequently the *meaning* of the data.

Brush = (B_{action, environ}) 23.
action: from A/R
environ: environmental registers at time T_a

A sonar sensor returns a specific distance reading which corresponds to an area that is cone-shaped and has an object at the far end of the cone, specified by the distance. A laser range sensor gives a more precise range reading and corresponds to a line segment of a given length. A system command, such as a reboot or self-test would paint nothing onto the map and is indicated with a NULL (\emptyset) brush element. Each robot system that can generate an event, and thus an MDS can be partitioned into one of two classes. The first class is those events whose actions and reactions are contained entirely within the robot. The second class is those events whose action and reaction require interaction with the environment outside of the robot such as a sonar sensor emitting sound waves that travel through the atmosphere surrounding the robot. The meanings of these types events are affected by the environment surrounding the robot. The sonar sensor is affected by the humidity of the atmosphere, most certainly the density of the medium

through which the sound wave must travel. It is thus important to be able to sense and record environmental conditions to make them available to the painting system when it constructs the map.

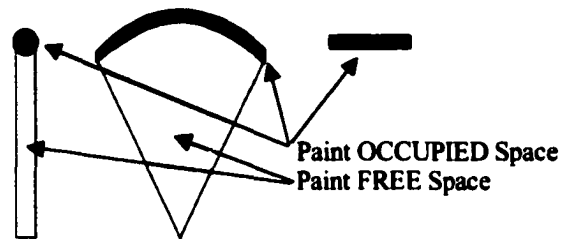


Figure 19. Brush Shapes

In Figure 19 we see graphical representations of the brush shapes used in our experimentation. The brushes depict the area to be painted with *occupied space* on the map after a sensor reading. All space between that brush face and the sensor are painted with *free space*. Left to right we have the laser ranging sensor, the sonar sensor and the contact sensor. The laser ranging sensor generates a single distance, and that results in the painting of a very small dot on the map. The sonar sensor generates an array of range readings, and each reading represents the distance to an object within a cone emitting from the sensor. Thus the sonar brush is an arc, representing the furthest extent of that cone at which point the first obstacle is encountered. The final sensor is the contact sensor, which detects a collision with an object. The space immediately in front of the motion, which is covered by the contact sensor, is painted occupied.

The amount and type of environmental sensors are dependent on mission requirements. Simple and wide tolerance missions would likely not require any such environmental surveying, however, specialized or critical missions may need such data. It is the goal that the MDL technology be as flexible and complete as possible to allow for the most complex mission imaginable even if the average mission would result in many of the features not being used. This flexibility is what gives MDL the ability to operate in the widely heterogeneous environments.

4.6 Correction Factor Element

The correction factor, or CorFac, element of the MDS is the part of the system that allows for correction of detected errors towards generating the most accurate map possible. The CorFac also allows for the corrections to be done via time dependent transforms dynamically during the mapping process

rather than a post-processing step. The correction factors are adjusted by the robot agent during mapping to cancel out as much of the discovered error as possible. Once the correction factors are adjusted, the entire MDS sequence can be parsed to repaint the map. This recomputed map reflects the corrective changes immediately. Additionally, the corrections can be quickly removed to return the map to its raw state if necessary. In a system with perfect sensors and actuators and without failure, all correction factors would be null (\emptyset). A robot agent performing a mapping task starts out with null correction factors as all data is assumed to be perfect unless errors are discovered via contradictory sensor data and only at that point would a correction factor be adjusted. The CorFac itself is a 4-tuple consisting of correction factors for all of the elements of the MDS except for itself.

$$\text{CorFac} = (CF_T, CF_P, CF_{AR}, CF_B) \quad 24.$$

CF_T : correction for time

CF_P : correction for position

CF_{AR} : correction for reaction

CF_B : correction for brush

The correction factor for time is to allow for the synchronization of the clocks among a distributed system of robot mapping agents if such a need were to arise. As clocks can drift, resynchronization would be indicated by an adjustment in the time correction factor within the MDL map. For most applications, it is anticipated that the time correction factor would remain \emptyset . The position correction factor is crucial to generating an accurate map. Error in the ability of the robot to know where it is to correctly register itself and thus place the gathered data into the map correctly is the most significant source of error in mobile robot generated mapping. Methods have been devised to assist a robot in locating itself but these methods tend to be impractical in unknown terrain [Talluri-92][Kabuka-87]. It is therefore very likely that a robot will lose track of its precise location in the coordinate system given a precise starting point and this will introduce error into the map as it is being constructed. Such error in self-location can be detected. Correcting for this through the position correction factor is key to the MDL method of error control. With the application of time dependent transforms to the MDS sequence over that part affected by a detected error in position, we can adjust for this error and produce a more precise map. The reaction correction factor addresses detected errors in the sensors and actuators themselves. This is

specifically designed to compensate for the systematic error that can be detected in the sensor and actuator systems. Stochastic error by its very nature is not predictable and will limit the effect it has on the mapping system over time. Finally, the brush correction factor allows for compensation in the brush due to deterioration from age, environment or other factors. If analysis were to show that a particular sensor was becoming very unreliable for a certain range of its design specifications, compensating via the brush correction factor would allow the data collected within dubious range to be filtered out. Perfectly working sensors requiring no correction factors would have a correction factor of \emptyset .

The flexibility of the MDL system is that it allows for heterogeneous robotic mapping systems to operate under very simple conditions where correction factors are likely to be of the form $\{\emptyset, CF_P, CF_{A/R}, \emptyset\}$ while still allowing the full capabilities if needed.

4.7 Error Detection and Correction in Historical Data

The primary motivation for the development of this technique was the necessity to construct more accurate and robust maps, and to accomplish this task, it was necessary to understand the reasons why maps constructed by mobile robots become inaccurate; we had to examine the sources of error. Error creeps into the maps we are constructing from several sources. Error is introduced via the physical sensors, via the tolerances of the gears in the drive system, from slippage of tracks or wheels on the ground, computational rounding off, and a wide variety of other sources. Some of these sources of error can be controlled or at least contained and some sources of error cannot. We define error, in the context of robotic map construction, as that additional component of the map, a sensor reading, etc., that deviates from the true value. For example, if we got a range reading from a sensor, r , we can express this value as follows:

$$r = R_{true} + error, \tag{25}$$

where R_{true} is the physically accurate reading and r is the reading returned by the sensor. What are the sources of the *error* component? We can partition the error component into two sections:

$$error = E_{STOCHASTIC} + E_{SYSTEMATIC}. \tag{26}$$

Stochastic error is that part of the equation beyond our control. It is the random variation in signals, return, and actuator function inherent to everything robotic component and system. Because of its

random nature, it is essentially noise on the signal we are trying to record and as such it limits itself. Since the effect of any error in mapping is potentially cumulative, the control and limitation of error is very important. Consider obtaining a robot's position by dead reckoning based on distance traveled. As one leg of trip is completed, error in position is incorporated into the position value. As further legs of the trip are completed, additional error is added on top of the previously incorporated error and the estimated position begins to drift more and more from the true position of the robot. In the case of stochastic error, the effect is limited due to the random nature of the error signal, which behaves as uniform distribution with mean of zero. This is not to say that there is zero effect from stochastic error through the mapping operation, but that the effect of this type of error is much less significant than the possible effect of systematic error.

The portion of the error equation that we must concern ourselves with is the systematic error; that error which is produced by some characteristic of the device producing the error. This systematic error can be caused by a component defect during operation, a manufacturing problem, environmental influences, and a host of other sources. Systematic error will accumulate in effect on the map data since, by its nature, it has a non-zero mean. Sequential readings and incorporation of such readings compounds the systematic error component of the total error. The non-random nature of this error causes a drift as the systematic error component becomes a larger and larger percentage of the perceived value, say position of the robot. The following figure illustrates this point (Figure 20). Assume that the systematic error of the wheel encoder and drive system is such that distance traveled is reported back as 10% short of the actual distance traveled. The robot actually travels 10% farther than it intended on each leg. As the robot makes a series of movements, the systematic error in the position will lead to an interesting result.

The effect of the systematic error in this simple illustration does allow our robot to return to the starting point as it intended since we assumed that there was no error in the turning operation, however it is clear that the path traversed is significantly different. Any sensor data collected along the journey would have been located in the incorrect places on the map being constructed and thus the map would see significant error in that data collected at the bottom of the journey (leg d).

Table 7. Effect of Error on Distance Traveled

Segment	Intended/Assumed Distance Traveled	Actual Distance Traveled
a	10m	11m
b	5m	5.5m
c	5m	5.5m
d	5m	5.5m
e	15m	16.5m
f	10m	11m

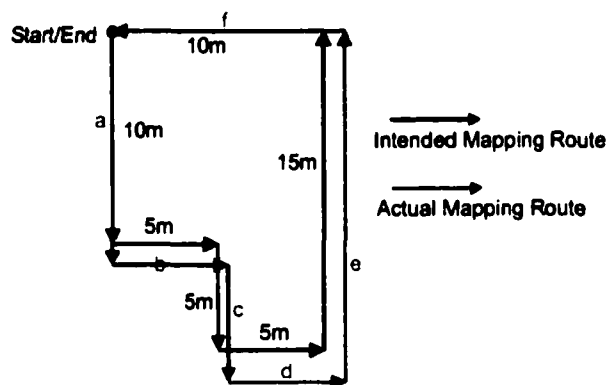


Figure 20. The Effect of Systematic Error

The effect of systematic error in a distributed and heterogeneous network of mobile robots is complicated by the fact that the mapping agents share mapping information and we can end up with systematic error components which result from readings taken by sensors other than those on the robot. Realizing that systematic error is a significant problem in mobile robotic mapping and that we may be able to control the effect it has on the map being constructed, we must first find a way to detect the presence of the error. Only if we can detect the error, can we isolate it and try to correct for its effect. Techniques which can be used to detect error are comparing theoretical locations with the actual maps constructed, using known reference points, triangulation from beacons and other methods. For the purposes of this work, let us assume that we have detected that there is an error our mapping data, such as the current robot position, and we would like to now treat the map such as to correct for the detected error. The mechanism by which this is accomplished in MDL is through the Correction Factor (CorFac) of the statements in the map description.

In an ideal map, there would be no error and we would produce a perfect replica of the real world on our map. The CorFac would be NULL (or zero) in this case, as no correction to the data is needed. In the real world, we would see error accumulating in the map as the robot mapping agent roams the unknown terrain taking sensor readings and incorporating those readings into the map via an MDS. Once an error in the map is detected, we can treat the MDS sequence to correct for the detected error. The correction is accomplished through the use of time dependent transforms which are applied to a subsection of the total MDS sequence. This time dependent transform (TDT) computes the CorFac components of each of the statements in the sequence based on assumptions we can make about the nature and effect of the error source. The goal is to create a map which has an acceptable level of error in it. This map is not going to be free of error but if we can eliminate any detectable error in the map, we have effectively constructed the best map possible within the technological limitations of our robotic system. MDL allows for the application of a TDT to the mapping data contained in the sequence of MDS which can correct for the detected error. When the MDL is again parsed to paint the map, a more precise map is constructed.

The form of the time dependent transform (TDT) is significant to the effectiveness of the correction factors in canceling the error in each MDS. Figure 21 illustrates two possible forms a TDT could take over the length of the segment of the map being corrected. The linear TDT indicates that the full correction factor, which compensates for the entire detected error, is applied to the last MDS entry and a correction factor of 0 is applied to the very first MDL entry (where we assume the data such as position is accurate). Along the way from the beginning to the last entry, we proportionally scale up the amount of correction over the entire sequence of MDL entries in the block we are correcting. This is one of the simplest types of correction we can make as we only need to count the number of MDL entries in the block we are correcting and then step through, adjusting the correction factor at each MDS. If we know something about the nature of the terrain over which we moved, then a non-linear TDT may be more appropriate. Knowing we transitioned from grass to gravel, for example, could indicate that much more wheel slippage and thus position error, was added in the gravel portion of the journey than on the grass portion of the journey. We could thus weight the correction amount more towards a particular segment of the MDL block, such as with a nonlinear TDT.

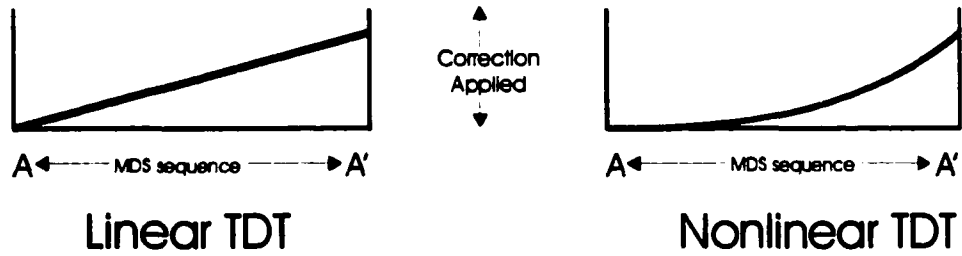


Figure 21. Linear and Nonlinear TDT

The linear TDT method does have some drawbacks. In particular, we are applying incrementally different correction factors to adjacent MDL entries when it may be the case that the adjacent MDS' should have the exact same correction factor. For example, if the two adjacent MDL entries are both used to fire a sensor and record the result and no movement or change in the robots configuration has taken place between them, then the correction for position in those two MDS should be identical.

The TDT we propose is to apply incremental correction only after events in the MDL that would cause changes in the correctness of the values. If we are correcting the position of the robot, we will only change the amount of correction applied from one MDS to another MDS if we transition over an MDS that involves movement of the robot. A sequence of sensor operations would not move the robot and result in the correction factors for that segment of the MDL block being all identical. The result is a staircase TDT keyed to the movement operations. We can compute the increment size, the step of correction applied to throughout the MDL block by:

$$Step_x = \frac{\Delta x}{NM} \tag{27}$$

$$Step_y = \frac{\Delta y}{NM}$$

where NM is the number of movement operations in the block of MDS from MDS_{first} to MDS_{last} , and Δx is the detected error's x component and Δy is the detected error's y component. Utilizing this computed step size we can formulate the computation of the correction factors to be applied at each statement in the block of the map description being corrected. The computation of the actual correction factor (CorFac) is then accomplished by:

$$CorFac_{Sx} = \Delta x * \left(\frac{S}{NM} \right)$$

$$CorFac_{Sy} = \Delta y * \left(\frac{S}{NM} \right)$$

If (MDS_{Sx} is movement) then S = S + 1

where S is set to 0 at the beginning of the block. We increase the correction factor after each movement of the robot and as a result have the same CorFac applied to MDS which should have no difference in terms of error of position. The resulting TDT would resemble Figure 22. This TDT improves the intuitive matching of sources of error, movement in this case, to the amount of correction factor applied to the various parts of the MDL sequence describing the map area in question. This TDT would work well if the majority of the error came from drift or wheel slippage at the starting or stopping point of a movement such as acceleration or braking only.

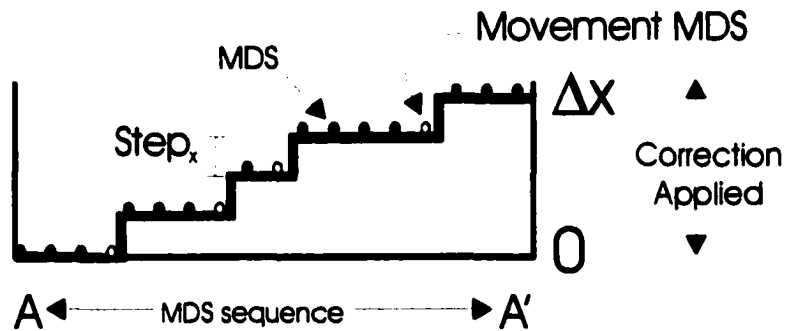


Figure 22. Staircase TDT Applies Correction More Intelligently.

If we assume we are getting significant drift in position resulting from wheel slippage and wheel encoder tolerances and other *in-motion* related sources, then we would be better served to associate the degree of correction applied with the distance moved in the motion operation. The TDT described in Figure 22 treats each robot motion identically, be it a move of 1 unit or a move of 500 units. We can define a new TDT that scales the correction factors applied to each of the movement segments to the distance moved within those motion operations. As we assume the motion itself introduces significant drift in position estimate, the segments of longer motion would contribute more error and thus be subject to a larger share of the correction. These correction factors are defined by the following equation.

$$CorFac_{S_x} = \Delta x * \left(\frac{SD_i}{D} \right)$$

$$CorFac_{S_y} = \Delta y * \left(\frac{SD_i}{D} \right)$$

If (MDS_{S_x} is movement) then $SD_i =$ distance moved in MDS_{S_x}

D is the total distance moved over MDS_{first} to MDS_{last} . SD_i is the distance moved in MDS_i where i goes from 0 to NM . If we take the same 4 movements depicted in the simplified staircase TDT and assign the movements a distance moved of 1,1,4, and 2 in sequence, then the TDT would be as shown in Figure 23.

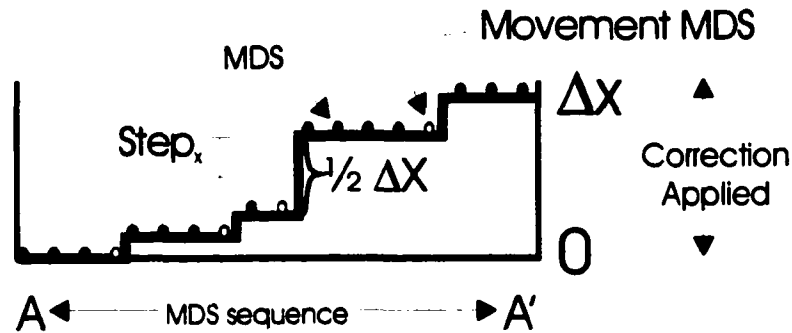


Figure 23. Distance Scaled Staircase TDT.

4.8 Cooperative and Distributed Map Building

In the following section, we will illustrate by example the map construction, error detection, correction via a TDT and global map construction. Before we do so, we must treat the final phase of our distributed robotic mapping system. The system consists of a heterogeneous network of mobile robots, each independently building a global map from the sensor data it collects. As was shown in Figure 15, these local maps are periodically broadcast and thus shared with the other robot agents. These collected local maps are then combined to produce a global map. Since the robots are heterogeneous, sensors and their characteristics are not known across the distributed system. While it would be possible to inform each robot about the necessary parameters needed to implement the brush functions and transmit the brush functions along with the MDL sequence as a copy of the local map, we will transmit the painted local maps with reference information so that they can be combined with simple data fusion techniques. In our case, we will use bitmap representations of the map to simplify the data interchange process by requiring no

parsing at the receiving end and making the data stream more robust. With a segment of the MDL corrupted during transmission, the meaning of all following MDS elements is meaningless because their meaning relative to the coordinate system would be in question as a position determining MDS such as a movement command may have been lost. With a bitmap representation, the corrupt section of the map is the only part affected and subsequent data in the transmission can still be reliably used. Once the local maps from all robots in the system have been received, each robot assembles its global map from these sub-maps. It is important to note that the information contained in the received local maps and the global map is never incorporated into any robot's local map. This prevents the spreading of erroneous data beyond the affected robot and only influences that portion of the global map visited by the affected robot. In a heterogeneous system such as this is, such data isolation is critical and allows for badly distorted or inaccurate data to be removed from the global map if need be. Robot mapping agents may use the global maps to guide them in exploring new regions but they do not reference the data in any way in constructing their local maps. Figure 24 illustrates this isolation of data, the flow of mapping information and the tasks associated with each map.

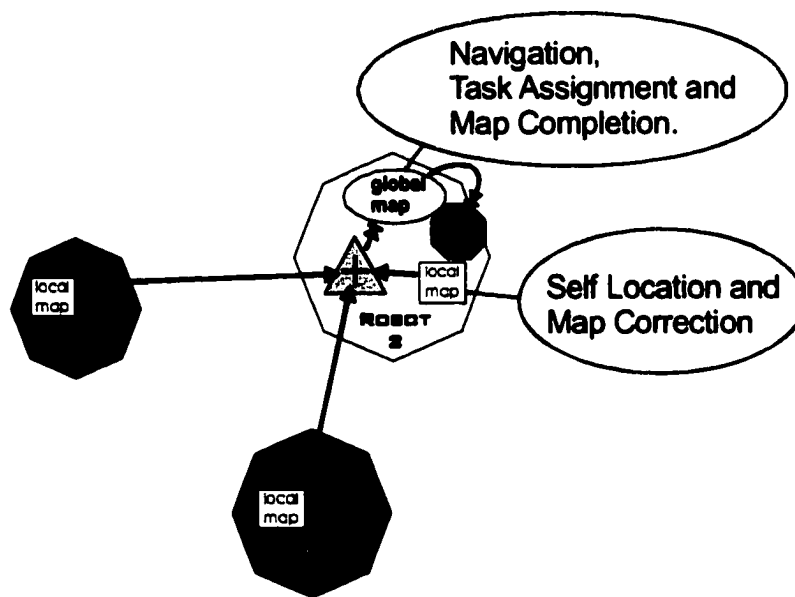


Figure 24. Map Data Isolation

The maps from robots 1 and 3 are combined with the local map from robot 2 to form the current local map on robot 2. The same process takes place in parallel on robots 1 and 3 to generate their global maps. If the map data from robot 3 was corrupted from a failed sensor or other undetected error or during transmission, this data would have no effect on what robot 1 sends to robot 2 and would only affect that part of the global map on robot 2 that was explored by robot 3. If the problem with the transmission or data from robot 3 is corrected in the future then at that time, the global maps would immediately reflect the correction without any memory of the erroneous data used in the past. Likewise, if robot 3 is decided to be defective, it can be eliminated from consideration by ignoring its data and the global map would reflect that change at the next global map update. We will now go through an complete example of the MDL based mapping methodology to illustrate how the system works and the benefits it contains.

4.9 An Example Illustration of the MDL Paradigm

To demonstrate both the problem and an application of the solution provided through the use of MDL as a map storage methodology, let us run through a simple example of the theories and methods developed so far. We will have a single robot traversing an area and navigating around a rectangle, returning to a location near its starting point. We will see how error in both position and rotation data can affect the constructed map and how such error conditions can be detected. Finally, we will see an example of one type of time dependent transform (TDT) that can be used to correct the mapping information by adjusting the correction factors (CorFac) for the MDS sequence. The corrections we compute are for the location portion of the position element of an MDS only; it allows us to simplify the example. Below is a sequence of steps the robot navigated to complete the journey and the collected data at each point. Some intermediate steps to complete a total map were omitted but could easily be filled in without affecting the outcome.

The following figures (Figure 25, Figure 26) show the mapping operation through seven steps numbered 0 through 6. Each diagram contains a view of the environment below which we see the local map the robot has constructed. In a heterogeneous, distributed robot system, this example would be played out multiple times in parallel, each robot independently building such local maps and performing corrections. Those local maps would then be shared and combined as indicated in Figure 15 to generate

the global maps. Examining a single robot allows us to illustrate the MDL methodology more clearly. Below the local maps we see a summary of the action/reaction (A/R) pair elements of each of the MDL statements produced as the robot navigates the scene. It is the A/R elements that are of interest as they contain the significant data that indicates the effect of the systematic error introduced. Additionally, the robot's position is shown relative to the *intended position* the robot assumes it has reached; systematic and stochastic error are responsible for this deviation between *intended* and *actual* positions.

As we follow the robot through its moves, we see that it generates an error in position in step 2 which carries over to step 3 as well as another in step 4. An error in rotation occurs in step 3 as well. As stated, there were some intermediate steps left out, which result in the gaps in the sides, these steps could be included without affecting the outcome of this example but were omitted to reduce the number of steps to those of particular interest.

We can see from the final map that the robot has completed its navigation around the exterior and arrived near the starting location. Notice that the points corresponding to the upper left corner, **A'** and **A**, are not in the same location on the map. Using simple geometrical knowledge of closed shapes and the planned and executed path, the robot can reason that these two points indeed represent the same point in the real world. The robot can thus detect that there is an error in the map it has generated. As we know that the two points are one and the same in the real world, we can compute the error in the position of point **A'** relative to the starting point **A**. It is assumed that the starting point is accurate. If dealing with simple x and y Cartesian coordinates, we can compute the error component as follows:

$$\Delta_x = A_x - A'_x \quad 30.$$

$$\Delta_y = A_y - A'_y \quad 31.$$

This allows us to define the error as

$$Error = (\Delta_x, \Delta_y) \quad 32.$$

We want to have *Error* be 0 ($\Delta_x = \Delta_y = 0$). It should be noted that *Error* contains both a systematic and a stochastic component but we are treating the stochastic component as limited in effect and negligible in comparison to the effect of the systematic error. If this assumption does not hold for a particular situation, then any attempt to correct error is futile as the uncontrollable stochastic error will

overwhelm any attempted correction and having no correction effort at all would likely result in a final map that was no better than one where correction was attempted.

We will now utilize a time dependent transform (TDT) on the MDS sequence to adjust the correction factors (CorFac) of the various statements resulting in an aligning of the two points **A** and **A'**. In our case, we will use a linear TDT that applies an equal distribution of correction to each of the MDL statements. An alternative TDT could be a logarithmic version (Figure 21) or a staircase version (Figure 22). The reason for choosing the linear TDT is for simplifying the example.

For the purposes of this example, depicted in Figure 25 and Figure 26, we will perform a correction only on the *position* element of the MDS sequences and only on the *location* part of that position. It is clear that *rotation* was also affected by error but the computation of the CorFac for *position: location* is sufficient to illustrate the methodology. The MDS sequence that we have for this mapping journey can be divided into sections for which position information stays fixed. The position information *could* have been corrupted as we transition from one section to the next. As we were not able to detect an error until reaching point **A'**, we have no way of knowing where any error may have been introduced into the system and thus we must treat the entire MDS sequence from the last assumed accurate point; the beginning in this case. After correction via the TDT, point **A'** is assumed accurate and subsequent corrections affect the MDS sequence beginning from **A'** until the next point an error is detected. Table 8 lists the correction factors applied to each of the elements in the MDS sequence over the locations that the robot visited during he mapping journey.

We can see that there were a total of 6 sections of position information. The introduction of error could have occurred at each of the 5 transitions. Using our linear TDT, we will correct for an equal amount of the error in each transition, thus setting the CorFac for all MDS elements in each of the six position sections to the same value. The amount of correction applied to the location part of the position element per transition is $(\Delta_x/5, \Delta_y/5)$. This is how we arrive at the CorFac values specified.

$$\text{CorFac.Position.Location (Position } P_i) = \{ (i-1) \Delta_x/5, (i-1) \Delta_y/5 \} \quad 33.$$

where i is the sequential position number from 1 to 6 as listed in Table 8.

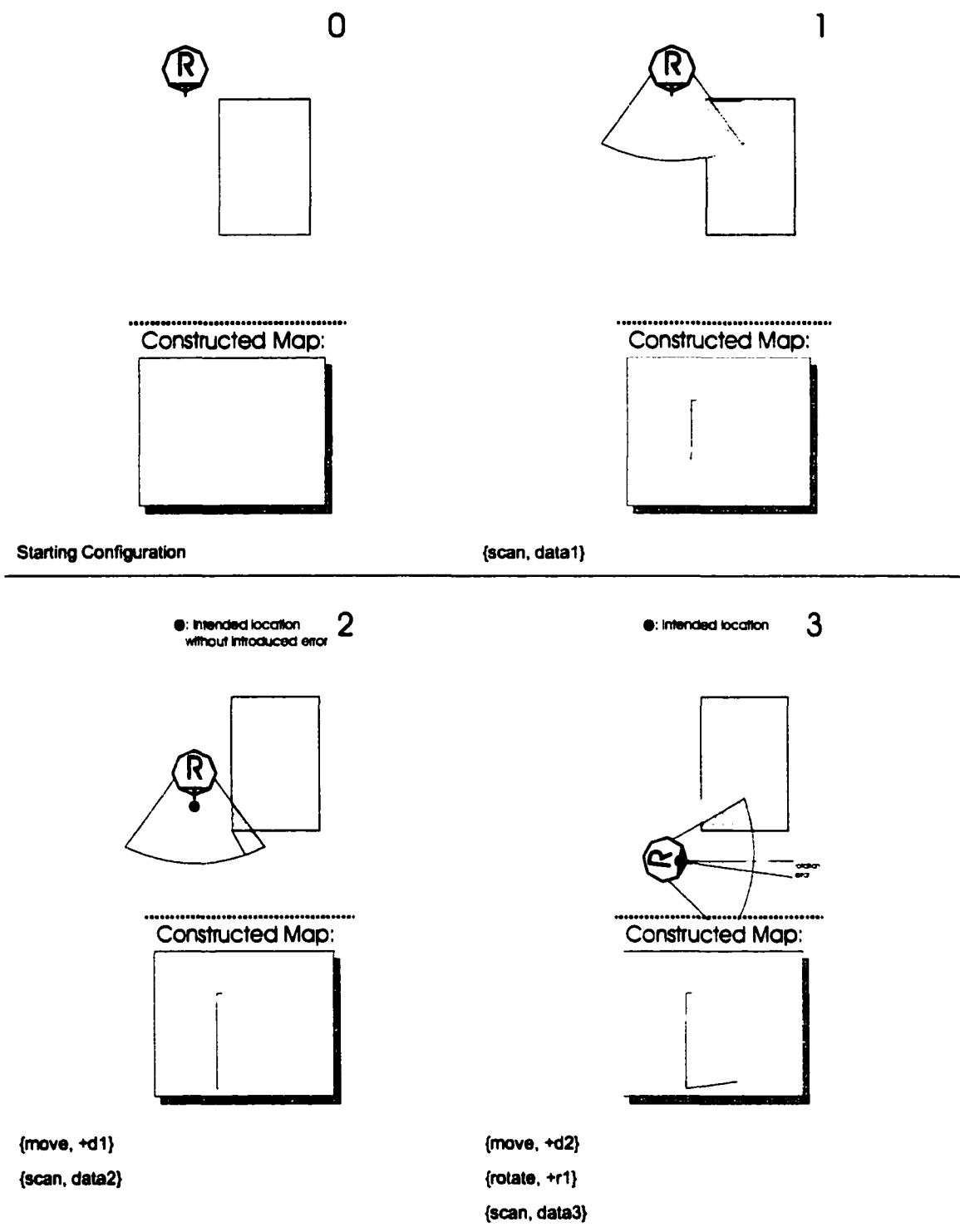
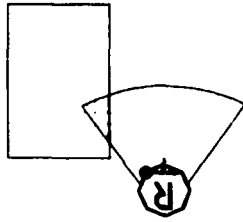
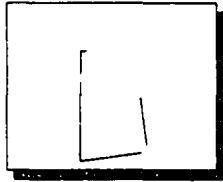


Figure 25. A Detailed Example, Part 1

●: Intended location 4

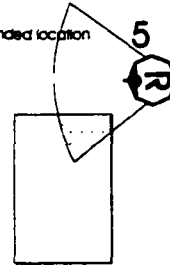


.....
Constructed Map:

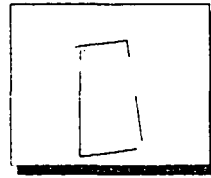


{move, +d3}
{rotate, +r2}
{scan, data4}

●: Intended location 5

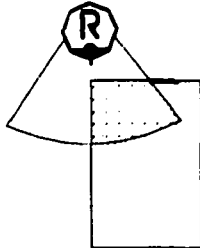


.....
Constructed Map:

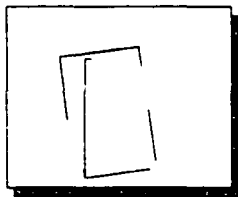


{move, +d4}
{rotate, +r3}
{scan, data5}

●: Intended location 6

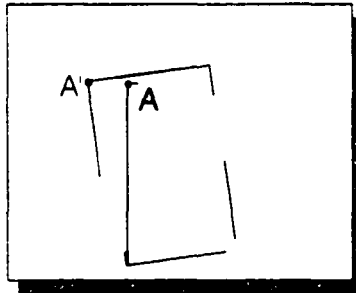


.....
Constructed Map:



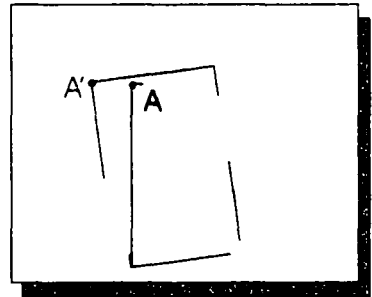
{move, +d5}
{rotate, +r4}
{scan, data6}

Constructed Map:



Final Map: Notice that points A and A' are the same location.

Constructed Map:



Final map overlay on the reference object

Figure 26. A Detailed Example, Part 2

The CorFac element illustrated is only the location portion of the position part of the overall correction factor of a single MDS. A full CorFac might look like

$$\{ \Delta_t, \{ (5\Delta_w/5, 5\Delta_w/5), (5\Delta_w/5), \emptyset \}, \{ \text{reaction} \}, \{ \text{brush} \} \} \quad 34.$$

where Δ_t is a possible correction to the clock, the rotation correction is also included, no configuration correction is set and a possible correction for the reaction (sensor data, maybe a fixed detected systematic bias) and brush (environmental influences) is included. Thus, a complete MDS element in the sequence, utilizing such a more complex correction factor, would take the form

$$\{ \text{Time}_a, \{ (x_a, y_a), (r_a) \}, \{ \text{"move +d}_a", \text{"+d}_a" \}, \{ \text{"sonar 1"} \}, \{ \Delta_{t_a}, \{ (5\Delta_w/5, 5\Delta_w/5), (5\Delta_w/5), \emptyset \}, \{ \text{reaction} \}, \{ \text{brush} \} \} \} \quad 35.$$

Table 8. Application of Time Dependent Transform

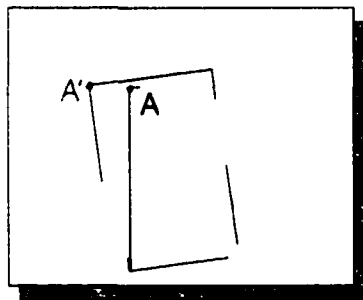
Position	MDS Sequence A/R pair	CorFac (Position: Location)
P1	scan	(0 Δ_w /5, 0 Δ_w /5)
	move +d1	(0 Δ_w /5, 0 Δ_w /5)
P2	scan	(1 Δ_w /5, 1 Δ_w /5)
	move +d2	(1 Δ_w /5, 1 Δ_w /5)
P3	rotate +r1	(2 Δ_w /5, 2 Δ_w /5)
	scan	(2 Δ_w /5, 2 Δ_w /5)
	move +d3	(2 Δ_w /5, 2 Δ_w /5)
P4	rotate +r2	(3 Δ_w /5, 3 Δ_w /5)
	scan	(3 Δ_w /5, 3 Δ_w /5)
	move +d4	(3 Δ_w /5, 3 Δ_w /5)
P5	rotate +r3	(4 Δ_w /5, 4 Δ_w /5)
	scan	(4 Δ_w /5, 4 Δ_w /5)
	move +d5	(4 Δ_w /5, 4 Δ_w /5)
P6	rotate +r4	(5 Δ_w /5, 5 Δ_w /5)
	scan	(5 Δ_w /5, 5 Δ_w /5)

Applying our linear TDT to generate our 6 correction factors and applying them to the constructed map from our simple example, we generate the corrected map shown in Figure 27. Two things are recognized from the comparison of the constructed and the corrected maps. First, we recognize that we

have indeed moved the point A' to the location of the true initial starting point, A . Second, we see that the corrected map does not match the reference object from which the map was constructed. This difference comes from three sources.

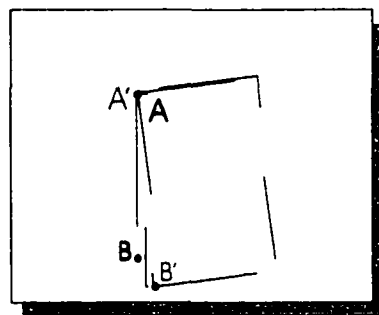
The first source is amount of correction applied to each of the legs. With our linear TDT, we applied an equal part of the total correction needed to move A' to A to each segment, without consideration to the amount of the total error that was contributed by travel on that particular leg of the mapping mission. If we review the information in Table 8, we see that the journey involved moves of various distances, those distances not necessarily being equal. If a movement generated a distance traveled that was greater than $1/5^{\text{th}}$ of the total distance traveled and we assume that some of the drift in position is a result of being in motion, then it reasons that the amount of correction applied to the sensor data collected on that leg should be greater than $1/5^{\text{th}}$ of the total correction for the whole journey.

Constructed Map:



Constructed map before correction with overlay of the reference object.

Corrected Map:



Constructed map after correction with overlay of the reference object.

Figure 27. Corrected Map for Example

The second source for the difference between the corrected map and the reference object is that the example also generated rotational errors, as illustrated in step 3 of Figure 25. However, there was no correction for rotation done by the TDT and thus no correction changes in the MDS sequence which describes our map. The rotational error is still included in the corrected map as it is painted.

The third source of difference between the corrected map and the reference object is error we could not detect with our method of error detection. We detected and computed the amount of error in

position by recognizing the shift in the upper left corner of the reference object which moved that corner in our map from position **A** to **A'** position. The mapping journey around the rectangle took the robot down the left side, but also back up the right side. Likewise it took the robot across the bottom and then across, in the opposite direction, the top. Assume that as the robot travels, the perceived distance traveled by the robot is less than the actual distance traveled by the robot. This might occur as a result of skidding by the robot when it stops motion. The wheels and wheel rotation encoder may well stop, but the robot itself may skid a bit further on the surface. Think of what happens in a car as you drive down a gravel road and then press the brakes suddenly. Although the wheels have stopped moving, the vehicle will continue forward for some distance. This is why the bottom edge of the constructed map sits below the bottom edge of the reference object that was mapped. This same effect would happen on each leg of the journey and so it would also occur as the robot travels up the other side of the object. When we got to our reference point to detect and compute the error in position, some of the error in the x-dimension and the y-dimension had cancelled itself out. We can compute the changes needed to match up position **A** to **A'**, however this is not based on the correction needed to match position **B** to **B'** (Figure 27). One correction of the x and y coordinates of the position of the robot as it proceeds through a complex path in the environment cannot be expected to accurately correct the map along all points of the path. Better correction results could likely be obtained by employing more reference points where error detection and thus correction takes place. However, it is not always possible to get correction points where you need them or want them. In our simple example, we have a robot mapping around a single object. The only location for reference that the robot knows is a single point – where it started. All mapping data is referenced relative to the location where the robot started its mapping journey. Once the journey was completed, we assumed our robot could reason that the corner it was seeing at the end of its mapping journey was indeed the same one it saw at the start of its mapping journey. This allowed the robot go compute a drift in position and make a correction for that drift. If additional points of reference were available in the environment from which the robot could obtain precise position information, then it would be possible to detect error and correct more frequently. We will utilize reference waypoints placed in the environment to obtain position readings from in our experiments in the next chapter.

5. Experimental Simulation of MDL and Map Correction

5.1 Conditions of the Simulation

5.1.1 World Model

The simulation environment provides a controlled test-bed for running algorithms and testing concepts where one can regulate the influences of the multitude of variables that would exist in a real world experiment. In the case of the robotic simulator that was written to test the MDL concept, we are able to enable just those effects we would like and disable those that we would like to eliminate. This allows us to see more clearly what affect each individual variable may have on the resulting performance of the system.

The environmental variables that can be simulated include physical error such as mechanical tolerances, communications errors such as message loss or corruption, stochastic error for any sensor or actuator, and the effect of wheel slippage during movement, just to name a few. For the sake of simplifying the analysis of the results, and to be able to categorize and organize our experimental results for presentation, we have limited the simulation of errors in our system. We simulate errors in rotation and movement of the robot, the wheel slippage and traction errors resulting from moving across imperfect surfaces, and noise in sensor readings which are dependent on the sensor type. Error induced into the mapping system causes the resultant map to be inaccurate with respect to the actual environmental condition it is supposed to represent. At the point of correcting for that error, it is irrelevant from which source or sources that error may have come. It matters only to what extent the error can be corrected.

5.1.2 Implementations and Systems

Robot Core Simulator and Algorithm Shell

In our simulator design, we implemented two major components, a robot simulator and a graphical user interface based simulation controller. The robot simulator was written in C and contains two major parts. The first part of the robot simulator is the core systems simulation, which contains the simulation code for all of the components on the robot and the data storage and access as well as communications abstraction. This part is responsible for generating the error distributions and injecting

those error distributions into the sensor readings and actuator movements, which are requested by the robot's algorithm running on the second half of the robot simulator. This part of the robot simulator acts as an abstraction layer to the algorithm processor for the robot, giving the algorithm access to basic functions to send and receive messages, rotate, move, fire sensors and receive their results. The robot simulator interacts with the true model of the environment to determine what data sets to provide back to the robot algorithm and adds the appropriate error where needed.

The second half of the robot simulator is a shell that runs in conjunction with the abstracted core routines in a loop and within which any particular robot algorithm can be executed as long as it can make use of the existing facilities provided by the robot abstraction simulator described above. The facilities provided to the robot algorithm include message reception, message broadcasting and addressing, storage of mapping information and reading of that mapping information from a predetermined grid of a 100x100 matrix that is 24 bits deep, and some utility functions.

In experiment one, a 'contour following' algorithm was written to cause the robot to trace the outside edge of a single object in the center of the 100x100 world and map it by using the simulated robotic sensors and wheels. The algorithm then implemented the MDL data storage approach by encoding all events into MDL statements and using the list of MDL statements to paint a local map whenever needed. This experiment allowed us to evaluate a limited implementation of our map storage paradigm and also evaluate the effectiveness of our TDT for applying correction.

The primary set of experiments ran a set of 2 and 3 robots over a simulated environment with 3 objects in them to test the performance of the system under multiple-agent and cooperative conditions. The world was again modeled on a 100x100 grid. A bounding wall surrounded the entire area to act as a container for marking the edge of the region to be mapped. Objects of arbitrary shape were placed inside the area at various locations and multiple mapping agents were then deployed, all using the same mapping algorithm, in fact, the same code, to construct a map of the terrain cooperatively. The agents each constructed a local map, as did the robot in experiment one. Periodically, those local maps were broadcast among the mapping agents and each constructed a global map from the combined data.

There is some *experimental* functionality provided by the core robotic simulator to assist in data analysis. The simulator provides a *goodness* measuring function which computes a reference number representing the difference between the local map generated by whichever algorithm is running on the simulator, and the true environment stored inside the core robot simulator. This provides us not only with visual evidence of the map but also some numerical data as well. These *coverage* values are computed via a difference summation between all of the pixels in the world model stored in the simulation and the map generated by the robot algorithm running in the simulator. The *coverage* is defined as follows:

$$Cov = \sum_{\forall i,j} \frac{|255 - Environment(i,j) - Map(i,j)|}{255} * R \quad 36.$$

where values of i and j are in the range 0 to the map width and 0 to the map height respectively. $Environment(i,j)$ is the true state of the world at grid coordinate (i,j) and has a value of 255 for occupied space and 0 for free space. $Map(i,j)$ is the corresponding recorded map information which has been constructed on the robot as a result of the sensor data gathered to this point. The map can be the local or the global map, depending on which map is being evaluated. R is a scaling factor.

Each pixel in the map can have a value from 0 to 255 so that that the simulator supports gray scale maps and could support algorithms that utilize *fading* or *occupancy probabilities*. In our algorithm, we used occupancy values for each pixel of empty (255) or occupied (0) in constructing our map. However, in the environmental model 0 indicates free space and 255 indicates occupied space. A perfectly drawn map would be an *image negative* of the environmental model. This is just the result of how our environmental model and our maps are created. We start with a blank grid of all 0 and fill in the desired information as 255 values. In the case of the environmental model, we paint in objects as collections of 255 value elements. For our map, we use the sensors to detect empty space up to the objects and paint in empty space as 255 values. Our scaling factor, R , was set to 2. Without a scaling factor, we would have a normalized coverage result that returns a number in the range of 0 to 1 for each pixel. We wanted to expand this range to 0 to 2 to avoid any issues related to rounding and integers within the computer software implementation where any value less than 1 would be rounded to 0. This results in the difference between the environmental model and the map model being scaled onto the range of 0 to 2. With our map

and environment fixed at a size of 100x100, the perfect match between environment and map would result in a coverage of 0, while a perfect mismatch between the environment and the map would generate a coverage of $R*Width*Height$, which is $2*100*100$ or 20,000. The bounds for the coverage statistic are thus 0 to 20,000.

Our analysis will be based on the changes in the *Cov* (coverage) statistic between the map without applying MDL corrections and with MDL corrections. The drawing of the map based on the use or non-use of the corrections is immediate as we only need to signal the map drawing function as to whether it needs to apply the corrections stored in the correction factors (*CorFac*) when it performs the painting. We are thus working with the exact same sensor and positioning data whether we use correction or not and can turn the correction *on* or *off* at will.

The verification of the correctness of the distribution generators was also performed experimentally by specifying the distribution desired and generating a log of the numbers generated and then analyzing that log to ensure it conforms to the desired distribution. The distribution used to generate the slippage due to traction problems on the wheels is based on a Gaussian distribution with a fixed mean and standard deviation. We limited the simulation of traction issues to only losing some motion due to traction and wheel slippage from acceleration and eliminated any *skidding* past the stop point from braking. Therefore, all of our distances moved would be equal to or less than the distance requested and reported. Each movement of the robot in the world is thus:

$$\text{Moved Distance} = \text{Requested Distance} - \epsilon \quad 37.$$

where ϵ is scaled from 0 to S and S is a fraction of the requested distance.

For example, if we specified S be 20% of the requested distance, then the error component, ϵ , is the distribution scaled to be from 0 to $0.2*Distance$. If the requested move was to move forward 10 units, then ϵ would be a random number from the distribution, scaled into the range from 0 to 2 and so the actual *Moved Distance* could be anything in the range from *Requested Distance* to *Requested Distance-2*. To achieve such a distribution, we took a basic Gaussian distribution, shown in Figure 28, and folded it over on its mean, mapping all of the points onto one side of the mean. This distribution is very similar to an

exponential distribution, but not quite the same. Figure 29 shows the density function of this Gaussian based distribution from one of the experimental runs.

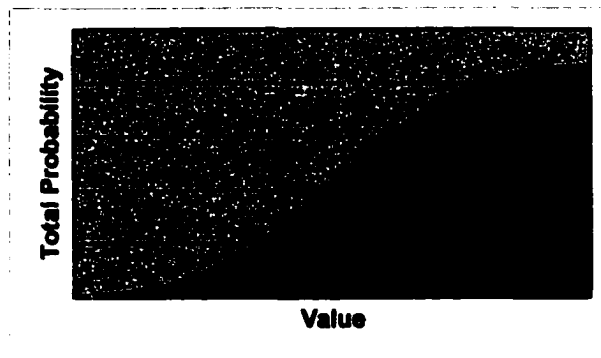


Figure 28. Gaussian Distribution Function

The Gaussian density is described by the equation:

$$f(y) = \frac{e^{-\frac{(y-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \quad 38.$$

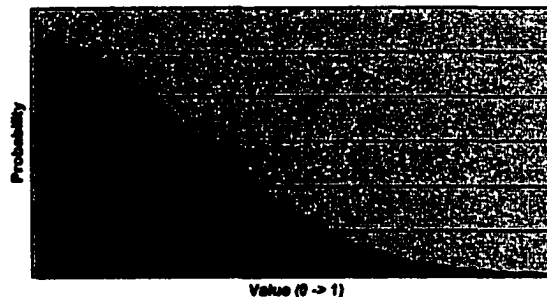


Figure 29. Gaussian Based Density Function for Error Simulation

Graphical User Interface – Simulation Controller

The other major component of the simulation system is the graphical user interface (GUI) and simulation controller. This part of the software simulator was written in Tcl/Tk, a language package of a core interpreted language, Tcl, and an extension package, Tk. This package is widely used and allows rapid, and more importantly, portable code to be written which generates interfaces in a GUI environment to allow for user input, output, display of graphics, etc. Tcl/Tk is available for a variety of platforms including Windows, Mac and various flavors of Unix. Our simulation utilized the Sun Solaris and

Windows NT 4.0 Tcl/Tk interpreters, version 8.0. The purpose of the GUI was to act as a front-end to the C code and allow the operator of the simulator to utilize their favorite *seat* as a controller and display device while being able to run the core robotic simulator on a more powerful or robust system such as a Unix server. The GUI interacts with the core robot simulator system described earlier via TCP/IP sockets and so is flexible enough to theoretically allow the robotic simulator to run in any location on the Internet, independent of where the GUI terminal is. We also wanted to allow robot processes to be run on any number of machines distributed over a lab, building, campus, city, country or even the world.

The simulation controller, dubbed UROSYS, for Universal Robot System Simulator, is run twice. The first instance is configured to be the *server*, which is the simulation controller. There is one and only one simulation controller for a simulation. A second instance of the UROSYS program is then configured as a robot GUI, which is then linked to a particular robot simulation already running. This allows for a robot to graphically display its mapping data and other information. All connections are managed through the single UROSYS server set up for the simulation run. Figure 30 shows a complete UROSYS desktop with two client robots connected to the server and a single GUI instance running for one of the connected agent processes (indicated in green in the client window). Figure 31 shows the UROSYS window used to configure the software to act as a server and then a GUI client.

As seen in Figure 32, UROSYS also configures the error distribution as well as the robot model. The robot model is a programmed set of algorithms that are implemented in the robot simulator as well as some variables that may need to be changed. The values set in the robot model are thus tightly coupled to the algorithms programmed into the robot simulator.

The robot configuration shown in Figure 32 allows the user to select from 4 algorithms. Algorithm 3 was used for experiment one for the experimental testing of the MDL mapping technique. Algorithm 4 was used for experiment two. Algorithms 1 and 2 were utilized only for UROSYS testing as well as robot testing to ensure the sensors and mapping functions operated correctly. Algorithms 1 and 2 just wander the map space collecting sensor data until the coverage statistic for the created map falls below the specified **COVERAGE** limit, at which point the robot would halt. Algorithm 1 just randomly turns when it bumps into an object using a uniform distribution to generate a new direction to try.



Figure 30. UROSYS Desktop

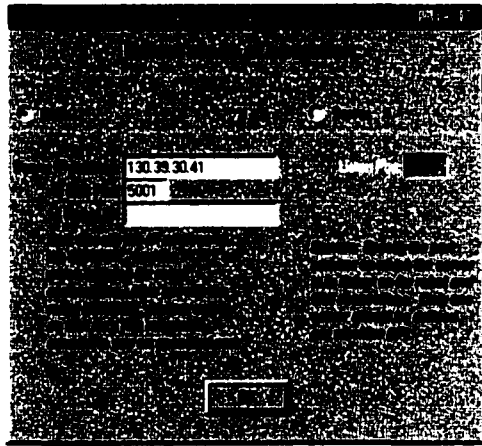


Figure 31. UROSYS Simulator Configuration

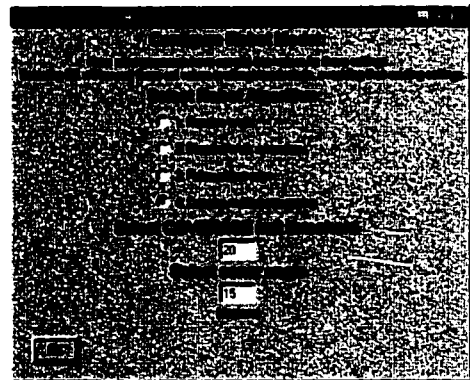
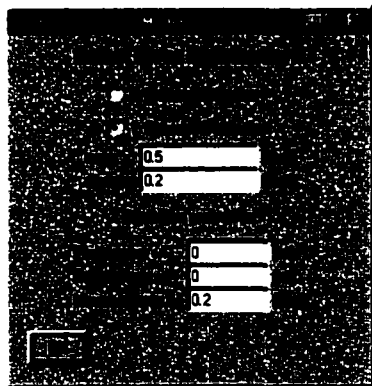


Figure 32. UROSYS Error and Robot Model Configuration

Algorithm 2 would use a more intelligent direction selection system, whereby the new direction was chosen by being *attracted* to the unmapped blank spaces of the map being constructed. The parameter, which did affect the experimentation, was the selection of the angle of the sonar sensors, which were programmed to be in a ring of 8 sensors, equally spaced around the circumference of the round robot. This sonar angle was not changed for any of the experiments run.

From Figure 32 we can also see the selection and configuration of the random number distributions that were utilized. The user could select from uniform or Gaussian, and then specify a mean and standard deviation for the Gaussian distribution. For our experimentation, we only utilized the Gaussian distribution to generate our error input. Additionally, the application and effect of the error distributions is controlled via three parameters. There is a fixed error added to any movement or rotation operation by the robot. By *fixed error* we mean that the range of the error is not dependent upon the amount of rotation or the distance moved. The range to which the distribution was scaled, before being added to the movement or rotational amount, is specified by the user. This magnitude is used to scale the error distribution generated random number, which is then added to the amount of a rotation or the amount of a move. Finally there is the movement error scale factor which is the percentage of the movement request which is used to scale the random distribution from zero to one to a more significant range. In Figure 32, we see a setting of 0.2, or 20%, indicating that the error distribution would be scaled to a value of 20% of the requested movement distance. That value would then be subtracted from the requested move distance to determine the actual distance the robot moved in the simulator. The algorithms running within the robot simulator of course assume that the full requested distance is moved.

5.2 Simulated Mapping of Objects

5.2.1 Experiment One

Our first experiments involved a simple task. A robot, always starting from the same location, was to circumnavigate its way around an object in a counter-clockwise fashion and create a map along the way. The resulting map would depict the shape of the object. We performed several experimental runs on each of several objects using several parameter configurations. The object shapes utilized were a potato, which included a concave as well as convex surface; a triangle, having more acute angles at the corners;

and a rectangle, representing the more commonly encountered man-made objects. The shapes can be seen in Figure 33.

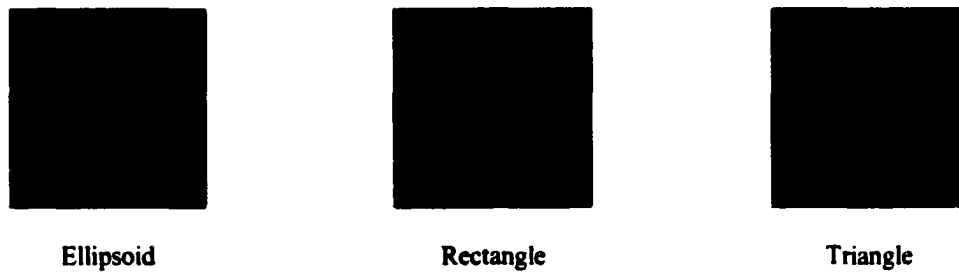


Figure 33. Shapes Mapped

The simulator generates four (4) error components of different types, one for each of the following simulated agent activities: rotation, movement, sonar sensor reading, and laser ranging reading. There are three (3) user defined variables set in the GUI that control these error component computations: position error, rotation error and movement error. Position error and rotation error are defined in absolute units and generate a Gaussian distribution over the range of their definition based on a distribution with a mean in the middle and a deviation of 0.2, ranging from 0 to 1. For example, if the *position error* were 2, then the error producing function generates a Gaussian distribution of random values ranging from -1 to +1 with a mean of 0. The sonar sensors utilize this error function. The laser range finder utilizes this function at $\frac{1}{2}$ the magnitude of the sonar sensor to signify the laser ranging technology is more precise than the sonar technology. The factor of $\frac{1}{2}$ is an arbitrary choice and not based on any particular hardware selection; it is merely designed to reflect the relative precision of the two sensors to one another. Thus we express the laser ranging and sonar sensor errors as follows:

$$\text{SonarError} = \text{PositionErrorRange} * \text{RandomGaussDistribution}() - (\text{PositionErrorRange}/2); \quad 39.$$

$$\text{LaserError} = (\text{PositionErrorRange} * \text{RandomGaussDistribution}() - (\text{PositionErrorRange}/2))/2;$$

The rotation error is defined likewise using the *rotation error* range as its basis:

$$\text{RotationError} = \text{RotationErrorRange} * \text{RandomGaussDistribution}() - (\text{RotationErrorRange} / 2); \quad 40.$$

For our experiments, we set the *rotation error range* and the *position error range* to either 0 or 2. For the case of 0, there is no rotation, laser or sonar error added to the signals or actions. For the case of 2, the

error added to the signals or actions ranges from -1 to $+1$ for the rotation or sonar operations and from $-\frac{1}{2}$ to $\frac{1}{2}$ for the laser ranging operation.

The overall movement error, that error added to the x and y coordinates as the robot moves through the simulated environment, is a function based on two parameters; *position error* and *movement error*. The position error portion of the overall movement error behaves just as described in the sonar sensor error description. Additionally, there is a *movement error* based component whose magnitude is a fraction of the distance traveled. The *movement error* defines what fraction that is. The overall movement error is defined as follows:

$$\text{OverallMovementError} = -(\text{PositionErrorRange} + (\text{MovementErrorRange} * \text{DistanceTraveled})) * \text{err} \quad 41.$$

where *err* is the Gaussian random distribution from 0 to 1 with mean $\frac{1}{2}$ and deviation 0.2 which has been folded in half and translated to generate half of a bell curve with a maximum probability at 0 and 0 probability at 0.5. Figure 29 illustrates this transformation.

The result is that the overall movement error is always a negative value. This overall movement error represents the loss of position accuracy due to traction loss or wheel slippage, which is accumulated at the start of movement and across the entire length of the movement operation. It is assumed in our simulations that the terrain is uniform, however the simulation framework does allow for the consideration of terrain changes and the effect that would have on computation of the overall movement error. Error added to position related to movement distance, resulting from physical tolerances in components and overshoot resulting from braking, are not included in our simulation model.

The simulated robot navigates around the object in a counter-clockwise fashion until it gets within a predetermined range of its starting coordinates. Along the way, it attempts to run parallel to the edges of the object, staying a fixed distance away from the object. The computation to rotate the robot parallel to the object face assumes that the robot is facing the object so that the front and front-left sonar sensors will pick up the object face to return range readings at points A and B, which lie in the center of the sonar cone at the respective range values returned by the sensors. If the coordinate of the points A and B are A_x, A_y and B_x, B_y respectively, then we can compute the angle of the orientation vector $V()$, depicted in Figure 34, as follows:

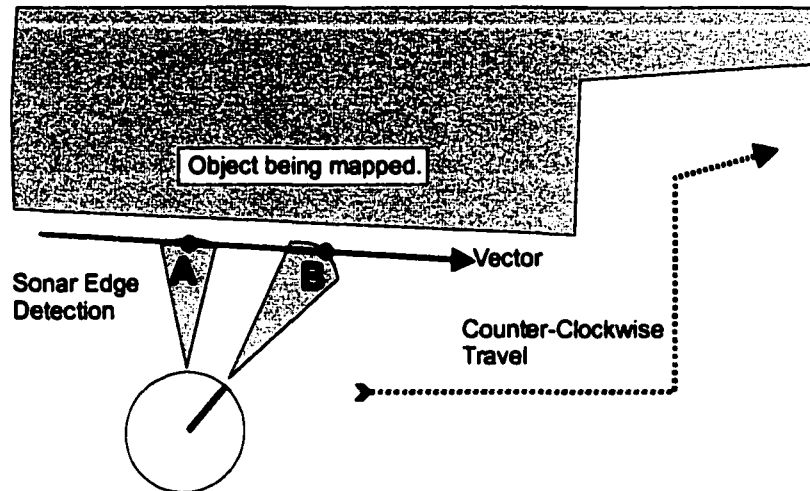


Figure 34. Obstacle Perimeter Tracing.

The following code computes the orientation angle, β , of the vector shown in Figure 34:

$$\Delta x = B_x - A_x$$

42.

$$\Delta y = B_y - A_y$$

$$d = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\beta = \cos^{-1}\left(\frac{\Delta x}{d}\right)$$

if ($\Delta y > 0$) then $\beta = -\beta$

After computing the orientation angle, β , of the vector, V , we compute the difference between the orientation angle and the robot's current orientation angle and rotate the difference to line up the robot parallel to the vector, V . Angles are relative to 0 degrees being due east. As the robot moves around the object and drifts further away from the object than specified (as read from a side mounted sonar sensor), the robot is rotated counter-clockwise, facing the object, backed up a small amount, and the orientation angle is recomputed and the robot again set parallel to the side of the object. If a front mounted sonar sensor indicates there is an object in front of the robot, then the robot immediately reacquires the vector, resulting in the robot rotating clockwise to follow the inside corner. If a sonar sensor on the left-front indicates there is no longer an object to the left, then the robot stops, rotates 90 degrees counter-clockwise, moves a small fixed distance, rotates another 90 degrees counter-clockwise and then reacquires the vector again. This allows it to turn around outside corners as well.

The algorithm allowed the robot to follow the outside perimeter of the object being mapped. It is assumed the objects are single closed objects. The algorithm ran successfully on the three objects we tested: ellipsoid, rectangle, and triangle. Upon reaching the location from where the robot started, it stops and ends the mapping operation.

5.2.2 Experiment Two

For our second set of experiments, we employed multiple mapping agents working in parallel on completing a global map of the environment. We utilized 2 and 3 robots, each running the same code to perform the mapping operation. We placed 3 arbitrarily shaped objects into the environment at locations to distribute them somewhat evenly over the environmental space. The mapping agents then performed a preliminary survey of their space wherein they placed intelligent waypoints that were required to remain within line of sight of at least one waypoint that had connectivity to the primary waypoint.

The primary waypoint is a single waypoint that was predetermined to be designated the origin of the world coordinate system. As each waypoint is placed, it determines its own coordinates in the world space by ranging and locating the primary waypoint, if it can see it, or by ranging and locating a secondary waypoint which has already determined its location from the primary waypoint or from some secondary waypoint. The waypoints remain fixed for the entire mapping mission and are collected by the mapping agents upon completion of the mission. The mapping agents can carry more than one waypoint for deployment in the environment. The waypoint deployment phase consists of simple object detection and avoidance. The robot agents fan out and deposit the waypoints in areas that are far apart or just before visibility to an existing waypoint is lost. A study of how best to place a fixed number of waypoints in an unknown environment is one area that is worth investigating beyond this dissertation. Figure 35 illustrates one world utilized for experiment two. The three objects are distributed around the world. The five light dots are the locations where the waypoints have been deposited.

Three mapping agents were utilized to construct a map of this environment and the results were analyzed similar to those obtained from experiment one. Figure 36 depicts the simulator desktop with two robot agents mapping the environment and a GUI interface open to one of the mapping agents to display its local map. The local map is displayed in the GUI window in the upper left of the figure and the main

server displays the environment model along with the location and orientation of all mapping agents in the window in the lower left corner of the figure. The window labeled 'Robot Clients' lists all mapping agents connected to the server and indicates which clients have an active GUI interface by the lighter (green) dot next to its identifier, 'R28591@ready' in this case.

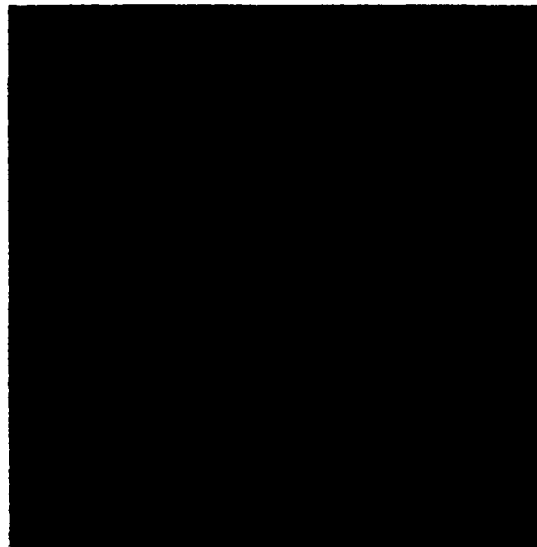


Figure 35. Multi-object Environment for Cooperative Mapping



Figure 36. Simulator GUI Workspace with Two Agents Mapping

5.3 Analysis and Interpretation of the Results

5.3.1 Experiment One

The object-mapping algorithm was run over 160 times with various settings of error effects and the results are compiled into the following tables and charts. The performance of the algorithm and the MDL method of data storage is as expected. The more error that is contained within the map being constructed, the greater the benefit obtained via the MDL correction factor adjustments. The three objects all resulted in similar results: there was a drastic improvement in the gains from utilizing the correction factors of MDL when a full error set was utilized in the simulation than when only a traction loss error was utilized. Additionally, performance was at its best when mapping a complex object such as the ellipsoid, with its varying curves, as compared to the regular geometric shapes which require little robot path correction to follow their contour.

The performance of the MDL system under the conditions of very limited error in the local map generated without any correction shows very minor gains if correction factors are utilized. The simple linear TDT scheme was utilized in experiment one. In fact, for the rectangle object, a very man-made object requiring few adjustments of the robot's trajectory during the mapping, we see that the application of MDL with only the traction losses resulted in a degradation of the quality of the resulting map. This appears to be attributed to a combination of very small amounts of error in the position at any given time coupled with the unit of measure and integer position limitations. The error values are on the scale of the smallest perceivable and representable distance and even a single unit (pixel) shift in the robot's position resulting from correction causes a degradation of the map quality. Once the error present in position from dead reckoning becomes larger we see an improvement in the maps after correction.

The interpretation of the results is based upon gains in the *coverage* value generated as a statistic within the robot simulation. The coverage is intended to indicate the degree of match between the true world model, which the agent is exploring, and the map that the agent constructs, as defined in Equation 22. This measure is by no means intended to be the best measure of mapping performance but it is a convenient one for computation and conveys the information desired. Additionally, the absolute percentages of gain are not of as much importance as is the comparison of the percentage gains between

runs of differing parameter settings. The gains are based on the number of pixels that differ between maps and are thus dependent on the size of the map as well as the number of objects in the map.

What can clearly be seen from the following figures and tables is that there is a definite improvement in the quality of the maps when correction was utilized in the presence of various sources of error in the mapping agents. The tables indicate the *shape* of the object mapped, the Position Changing / Rotation Changing error contribution and then list the movement error fractions used with the average gain in coverage over the set of simulated runs at that setting. We set the position and rotation fixed error rates to zero for one run, to only capture the effect of the traction related error due to movement as a fraction of the distance moved. We then set the position and rotation fixed errors to a scaled range of 0 to 2 pixels across which the gaussian distribution is scaled to thus produce a movement or rotation error within the range of ± 2 pixels. The results are tabulated in Table 9, Table 10, and Table 11 and charted in Figure 37, Figure 38, and Figure 39. In Figure 40, Figure 41, and Figure 42 we have charted the performance on an absolute scale where performance gains are shown in the number of raw pixel differences between the uncorrected and corrected maps.

We also analyze a normalized result (Figure 43, Figure 44, and Figure 45). We computed the absolute average change in the coverage values and normalized it by the number of pixels in the image. This result is defined by the following equation:

$$Gain_{norm} = \frac{\sum CovGain / n}{p} \quad 43.$$

where *CovGain* is the coverage gain for a single run, *n* is the number of runs in the set and *p* is the number of pixels in the image.

As we utilize 100x100 maps, the number of pixels is therefore fixed at 10,000. The larger the map would be, the greater resolution or detail could be captured over the same terrain area covered, and the more pixels would be contained within any given map. The minimal pixel size is also bound by feature size we want to capture and by robot size, however these criteria were not considered in choosing our map resolution of 100x100. The selected size is arbitrary.

Table 9. Rectangle Results

Rectangle

Pos/Rot=0/0

MovE	Avg Imp
0.150	-65.211
0.175	-71.778
0.200	-64.552
0.225	-28.801
0.250	-19.301

Pos/Rot=2/2

MovE	Avg Imp
0.150	41.396
0.175	40.802
0.200	42.851
0.225	49.101
0.250	31.966

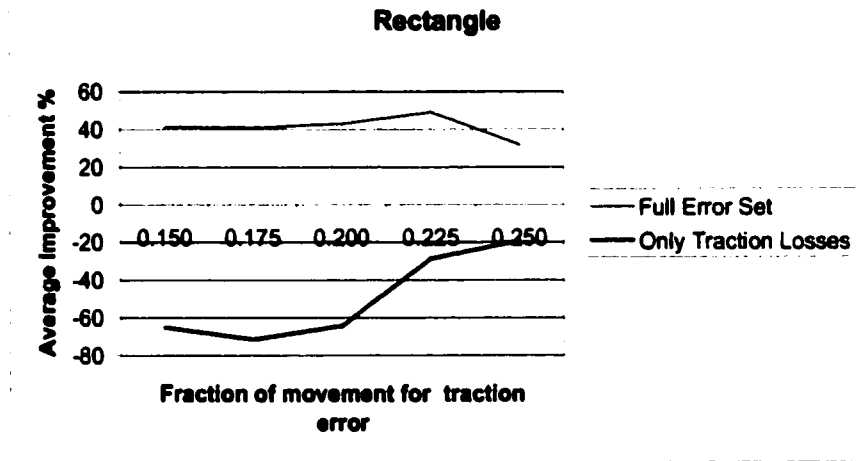


Figure 37. Rectangle Results

Table 10. Ellipsoid Results

Ellipsoid

Pos/Rot=0/0

MovE	Avg Imp
0.150	2.881
0.175	4.193
0.200	0.225
0.225	7.069
0.250	9.973

Pos/Rot=2/2

MovE	Avg Imp
0.150	29.928
0.175	18.248
0.200	20.612
0.225	16.828
0.250	18.877

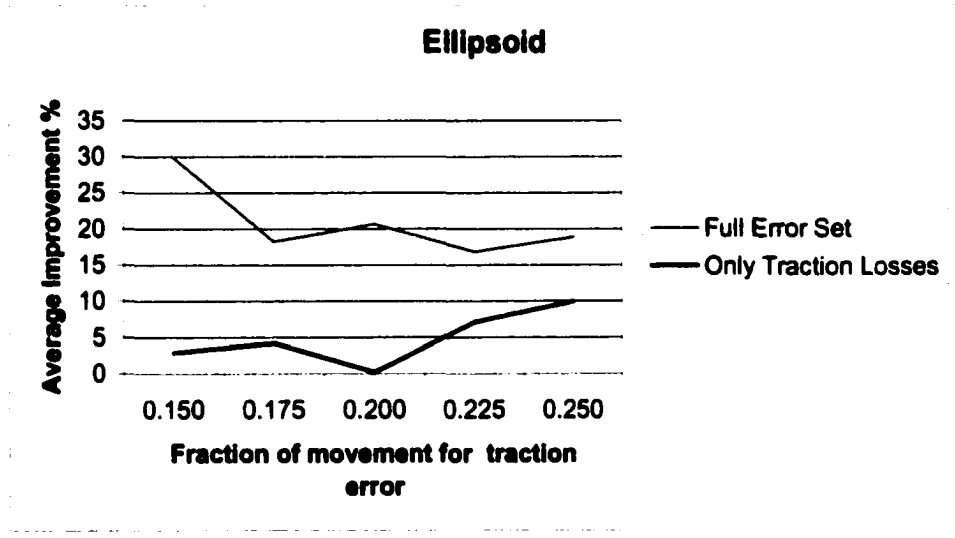


Figure 38. Ellipsoid Results

Table 11. Triangle Results

Triangle

Pos/Rot=0/0

MovE	Avg Imp
0.150	-4.274
0.175	1.882
0.200	1.307
0.225	7.979
0.250	12.949

Pos/Rot=2/2

MovE	Avg Imp
0.150	44.135
0.175	44.508
0.200	45.065
0.225	49.888
0.250	50.248

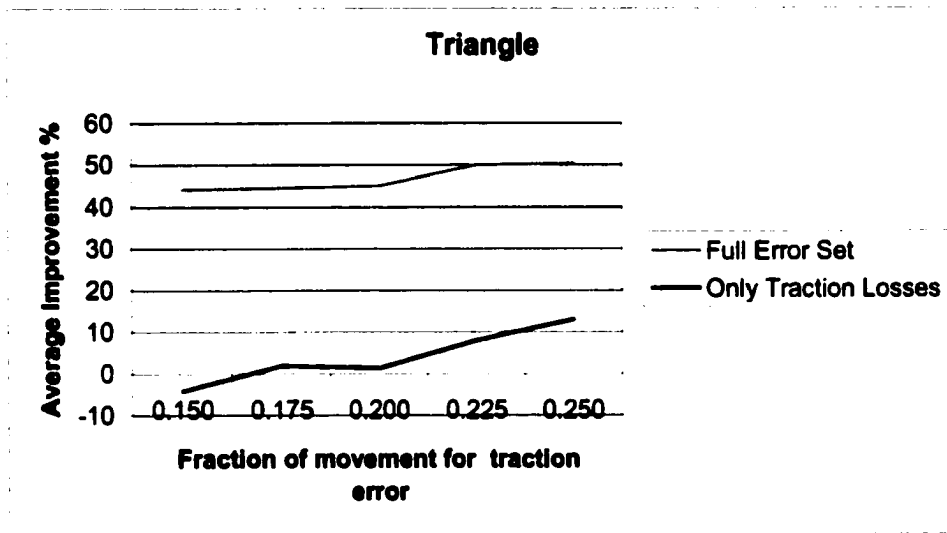


Figure 39. Triangle Results

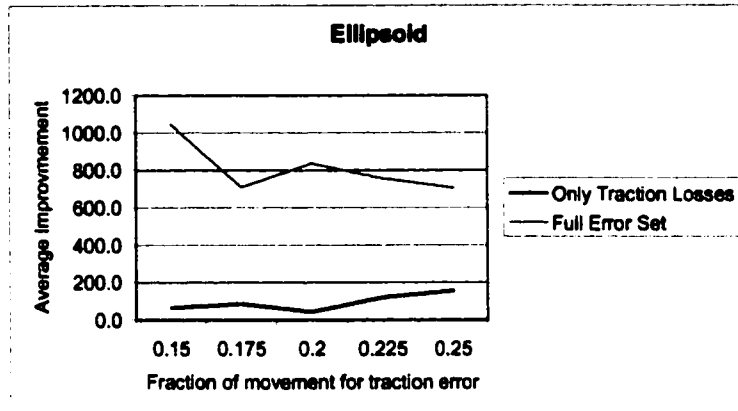


Figure 40. Absolute Gain in Ellipsoid

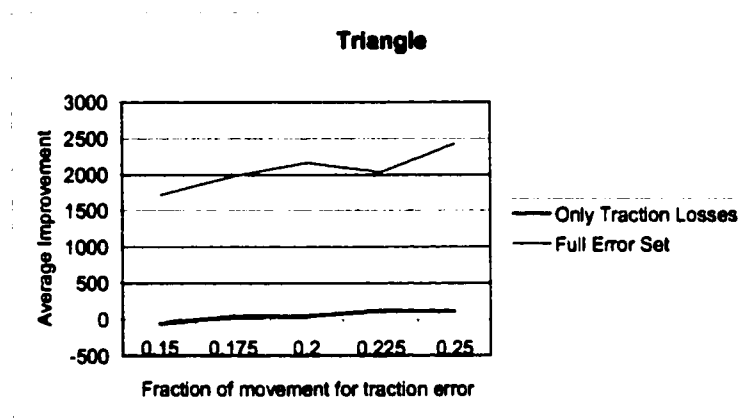


Figure 41. Absolute Gain in Triangle

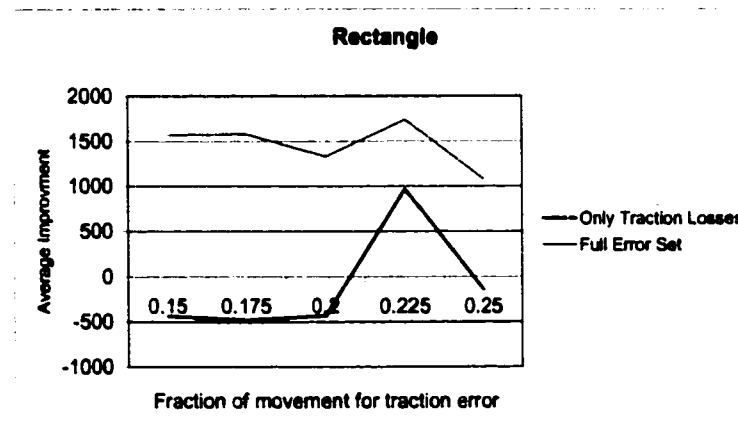


Figure 42. Absolute Gain in Rectangle

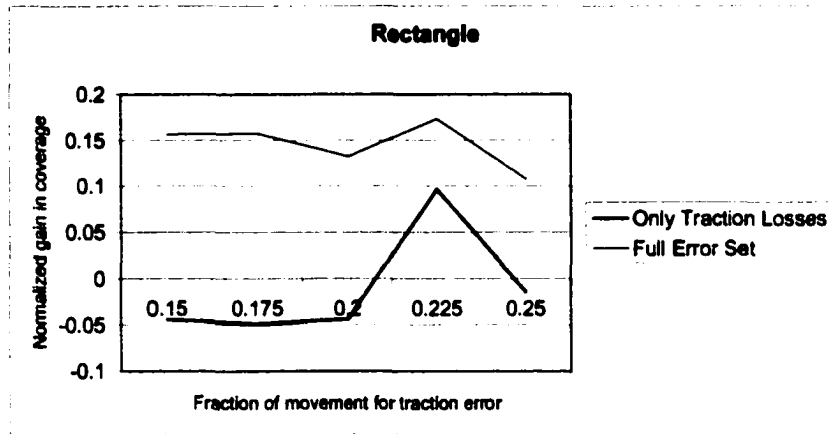


Figure 43. Normalized Gain in Rectangle

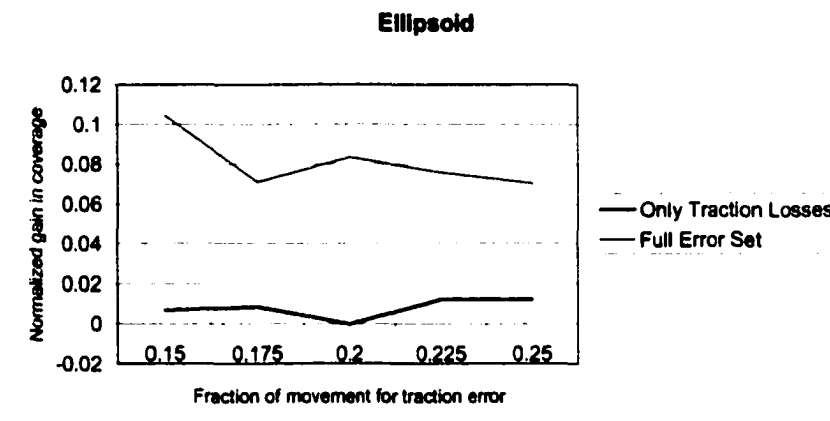


Figure 44. Normalized Gain for Ellipsoid

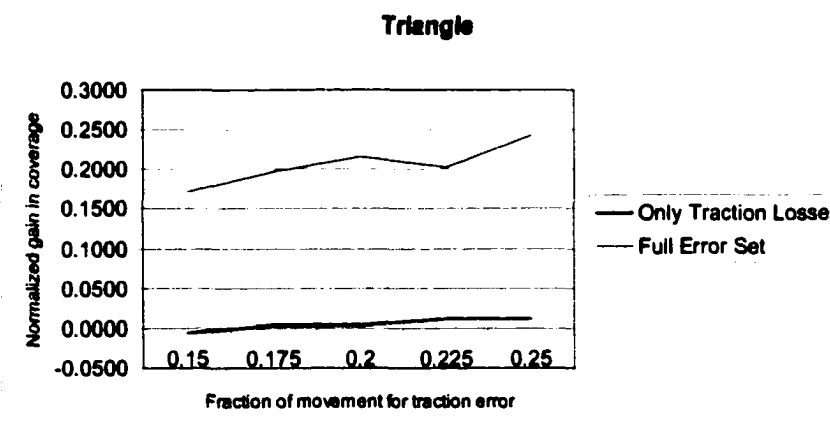


Figure 45. Normalized Gain for Triangle

The performance losses, particularly with regard to very simple objects with little induced error on the data such as is depicted in Figure 37, can be partially related to the implementation of the robot process simulator and how the data is stored internally. Within the core agent process, coordinates are recorded as integers. This decision was made early on in the development process to allow for simpler comparison between coordinate values. It was assumed that the world maps and constructed maps would be of such a magnitude in size that the direct link between integer coordinates and pixels in the grid based map would not be a problem. Once development proceeded far enough to test the simulator, it was discovered that the computational resources at our disposal would not allow for efficient simulation of maps that were represented by a grids of higher resolution such as sizes of 1,000 x 1,000 or larger. The area covered by such larger grids would contain information about the same region in space, but with greater detail. The memory resources and computational speed on the machines utilized for the simulation would have been severely tested. Six megabytes of memory would have been required for storing a single 1,000 x 1,000 map. Since each robot would require at least 4 map spaces, for a total of 24 megabytes of memory just for the map, plus memory for the rest of the code along with the operating system, it quickly became apparent that the speed at which such simulations would proceed would seriously affect our ability to complete the experiments within a reasonable time frame. It was decided to restrict the maps to 100 x 100 in size. The size of the grid used to represent the world is not a limit on the size of the environment explored, but determines the resolution of the world and the maps constructed. The use of variable resolution techniques such as those proposed by [Arleo-99] or other methods of compressing the grid information in regions of open spaces could reduce the cost in terms of memory if real robotic resources were fixed. Once preliminary results were obtained, it became apparent that size of the agents, a circle of radius 5 units, with a minimum detectable movement of 1 unit (pixel), being 20% of the size of the robot would be a problem. Movements smaller than a unit (pixel) would be rounded down and lost and this created a tendency to shift all data by one unit to the smaller side. A conversion of the data type for coordinates to floating point numbers was briefly considered. Such a change would allow the recording of much smaller changes in position that would be represented in the maps and their analysis. However, it became apparent that changing the coordinate data type from integer to floating point at this late stage

would require the rewriting of a significant portion of the nearly completed simulator and that this would delay the experimental runs significantly. As we were going to be testing the effectiveness of the MDL corrections over a wide range of error magnitudes, it was determined that the effect of the scale of the smallest represented quantity, the unit or pixel, versus the size of the entire world model, being 100 units or pixels wide, would not significantly affect our ability to interpret results. While it is clear there is an effect as a result of this limitation, that effect does not detract from the trends in the data we have analyzed that show a definite gain in the application of the MDL paradigm in situations where error sources are plentiful. In a real world deployment of such a system, the simulator memory requirements as well as the world model representation would not be needed in the memory of the robot itself. Likewise, the robot agents can be designed and built to the specifications needed rather than being limited to the construction of older systems on which the simulator was run.

In examining the normalized gains for the three objects mapped, as depicted in Figure 43, Figure 44, and Figure 45, we see that the performance of the correction is less sensitive to changes in traction related errors than it appears in the percentage change graphs. This is particularly visible in the ellipsoid results depicted in Figure 44 compared with Figure 38. It is still clear that there is significant gain in the quality of the final map under conditions of more error sources or more error magnitude as compared to limited error sources and limited error magnitude.

5.3.2 Experiment Two

For our second set of experiments, we performed mapping operations on an environment of multiple objects with multiple robots acting cooperatively. Multiple mapping agents were placed into the environment and proceeded simultaneously to map the space. Local maps were constructed by each robot similar to those of experiment one with the exception that each robot followed a space-filling algorithm to seek out and explore a section of the map. When a dead end in the space filling operation was reached, an agent would locate an unexplored region of the global map to determine where to explore next. Once the agent has navigated its way there, it would begin a new space filling exploration step into this unexplored region. When the global map was completed within a preset tolerance, mapping stopped and all agents halted at their locations.

Significant improvements of the maps constructed were observed, particularly when a more complex TDT was employed to apply correction. Improvements in the quality of the map under correction are shown in the following figures of some of the experimental runs. Figure 46 shows a completed global map after correction from one of the experimental runs. The image on the left is the completed map and the image on the right is the coverage map which depicts the differences between the global map and the actual environmental model. Areas of agreement are black in the coverage map, while areas of disagreement are light (green). With an exact match of the final global map to the environmental model we would have a totally black coverage map. Figure 47 depicts an uncorrected and a corrected global map with the uncorrected map on the left and the corrected map on the right. It is clear, when one takes into account the environment model utilized and depicted in Figure 48, that the corrected map on the right of Figure 47 more accurately portrays the locations and the shapes of the objects in the environment. The correction to the shape of the rightmost object is very apparent. From a qualitative viewpoint, the corrected map is the output one would like to obtain from a mapping mission. The darker, banana shaped marks on the maps are artifacts of the method used to locate and differentiate unexplored regions in the global maps.

Further qualitative results are given in the partial maps shown in Figure 49, Figure 50, and Figure 51, which depict some of the improvements obtained in local robot maps as a result of using various TDT methods to perform corrections on-line. Recall that a *linear TDT* uses a straight line increment in the amount of correction applied to each successive MDL element in the block of the map description being corrected. The *stepped TDT* applies correction by dividing the total amount of correction over the number of movement operations performed in the block being corrected and keeps the correction amount constant between movements. Starting from the assumption that the most significant source of error is from the movement of the robot and not from sensor actions, all map description elements that are generated at a fixed location receive the same correction factor. The *scaled stepped TDT* improves on the stepped version by scaling the amount of correction applied to any step by the portion of the total distance covered in movement within that segment of the block. In Figure 49 we see a partial local map, depicted as a coverage map, that was constructed and corrected with the basic linear TDT scheme. The uncorrected coverage is shown on the left while the corrected coverage is shown on the right. The improvement



Figure 46. Completed Global Map and Global Coverage



Figure 47. Uncorrected and Corrected Global Map



Figure 48. Environment Model

around the leftmost object is quite dramatic in this case. The coverage shown in Figure 50 is from a stepped TDT. The uncorrected map is on the left while the corrected map is on the right. It is not immediately apparent where the gain in Figure 50 is. If one focuses on the peak on the rightmost object in the maps in Figure 50 and the same object in Figure 49, one sees that the depiction of the peak is more even with the stepped TDT of Figure 50 than with the linear TDT of Figure 49. Figure 51 depicts the improvements in the coverage map when a scaled stepped TDT is used for applying correction factors to the map description. The corrected map is on the left here while the uncorrected map is on the right. The dramatic improvement in the matching of the object at the bottom and the one in the upper right is clear.

In addition to this qualitative analysis of results, let us now examine some quantitative differences between maps that were corrected or not as well as the affect of TDT selection.

The performance of the system on the simulator was similar to that seen in experiment one. The more error that was introduced into the motion and sensor operations, the more significant became the gains realized from correction. Since the task of mapping the space was shared among various robots, the local maps of individual mapping agents were only partially completed at the point the global map was completed. Analysis was again performed based on the coverage statistic as it pertains to the local maps of each robot, but rather than examining the percentage improvement, we examined the absolute number of pixel difference in the uncorrected coverage versus the corrected coverage. The results can be seen in Table 12 and Figure 52.

We can see that there is more significant gain resulting from map correction, as more error is present in the system. Even under only the limited error set of traction losses we obtained gains from the application of correction, as compared to the results from experiment one, where very small amounts of error resulted in some degradation in the quality of the maps constructed. This can be attributed to the greater availability of reference points in the way of the intelligent waypoint markers which were pre-positioned in the environment by the mapping agents as they were deployed. The result of having more waypoints is that correction of drift from dead reckoning is more frequent. As a result of the greater frequency of correction, more error is detectable than was the case in experiment one. In fact, some preliminary runs of experiment one with only a single waypoint resulted in rather poor performance from



Figure 49. Coverage Gains from Linear TDT



Figure 50. Coverage Gains from Stepped TDT

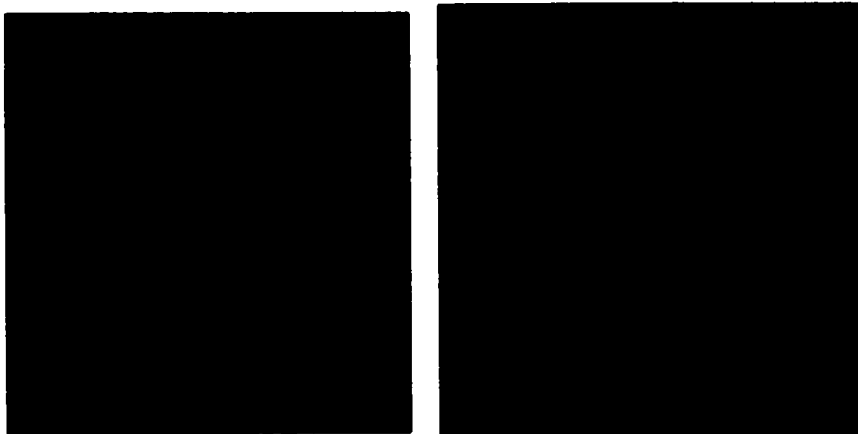


Figure 51. Coverage Gains from Scaled Stepped TDT

Table 12. 2 Robot, 3 Object Results

Movement Error	Traction Losses	Full Error Set
0.15	20.1	152.1
0.175	76.5	154.3
0.2	186.3	345.9
0.225	247.3	464.5
0.25	265.7	699.3

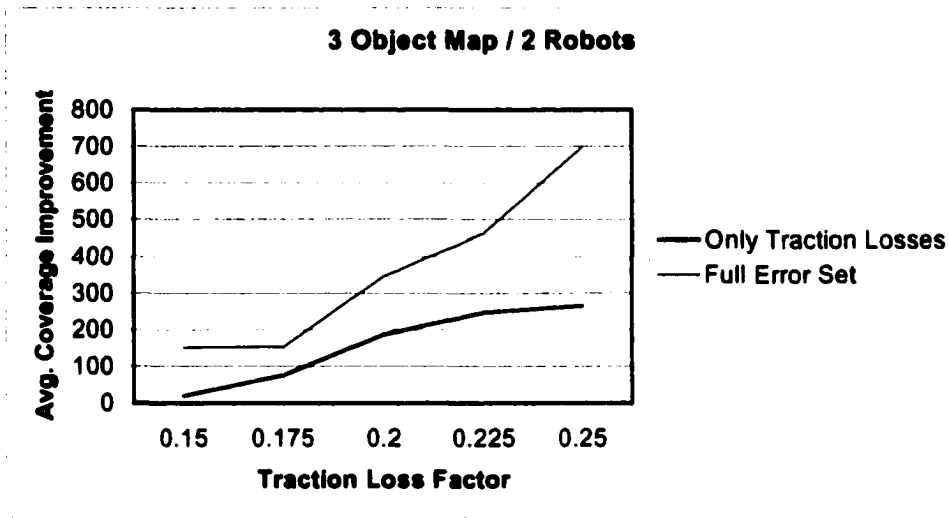


Figure 52. Results with 2 Robots and 3 Objects

the detectable error being greatly reduced as a result of the robot making a complete loop around an object. It is clear that if dead reckoning is to be used as a method of maintaining a global position reference, then the availability of some form of known and fixed landmarks, be they *a priori* or introduced, is essential for position location and correction.

Additionally we tested the performance of the three different time dependent transforms (TDT) discussed earlier under similar conditions. The linear demonstrated the most limited improvement. This was to be expected as the application of correction to each piece of the map description was without regard to where the error within any point of that segment of the map could have come from. The stepped TDT performed marginally better. The best performance was obtained by the scaled stepped TDT, which went further than the stepped TDT. The results of these experiments are seen in Table 13 and Figure 53.

Table 13. TDT Performance Results

Traction Loss Factor	Linear	Stepped	Scaled Stepped
0.15	152.1	141.4	198.2
0.175	154.3	201.1	253.9
0.2	345.9	358.6	434.7
0.225	464.5	557	679.7
0.25	699.3	731.5	859.5

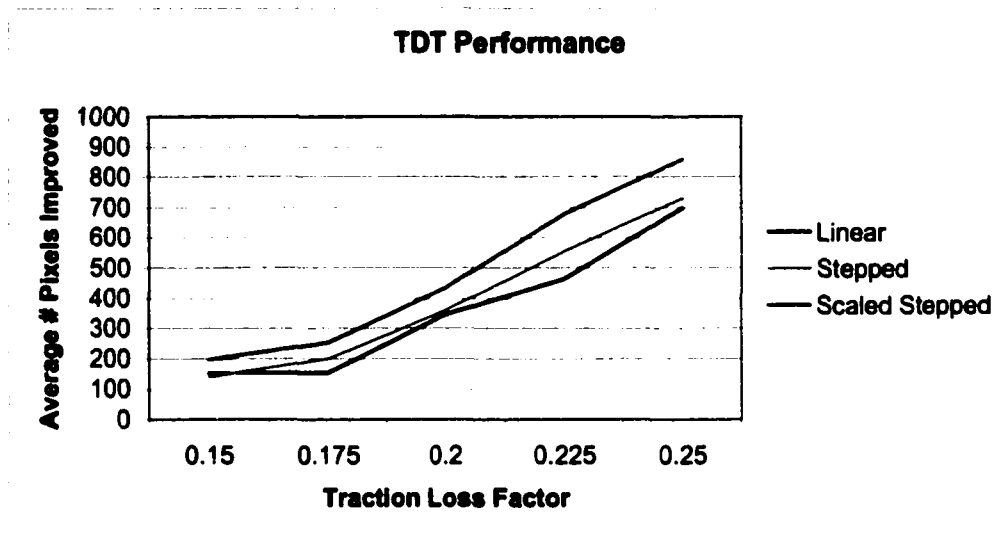


Figure 53. TDT Performance by Type

In addition, we can examine the normalized version of the results obtained based on the definition given in Equation 43 to compare with the normalized results given for the single robot mapping the edge of a single object (Figure 54 and Figure 43, Figure 44, Figure 45).

The resulting graph is identical to the non-normalized version other than the y-axis units but it allows us to compare with the single agent mapping results in Figure 43, Figure 44, and Figure 45. We see that the normalized gain for the three object and 2 robot mapping result is below the results from the three object types when a full error set is utilized. There appears to be less improvement when multiple robots are utilized however this is not the entire picture. The world for experiment two contains 5 reference waypoints from which agents can obtain position correction information while the single robot runs of experiment one only contained two such points. The robot in experiment one would spend much more time, on average, between corrections and thus accumulate much more error from dead reckoning. As we

have already seen, maps containing higher levels of detectable error will show a more significant improvement from the application of TDT correction in conjunction with the MDL system.

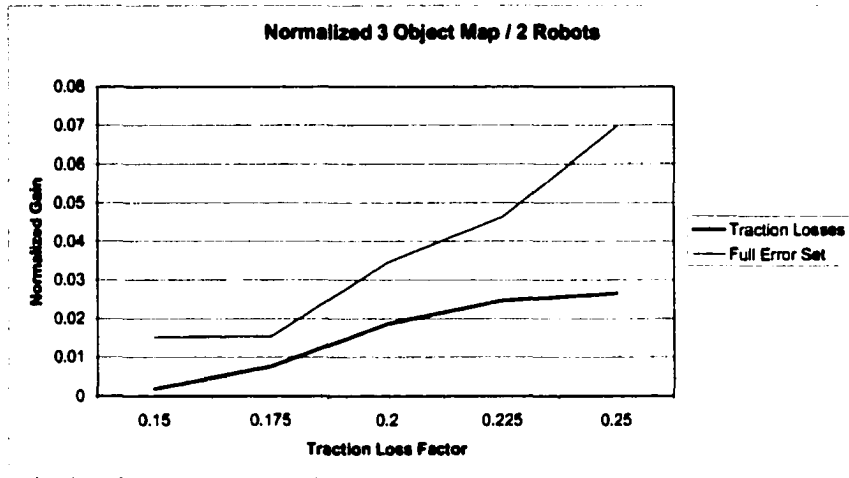


Figure 54. Normalized Results with 2 Robots and 3 Objects

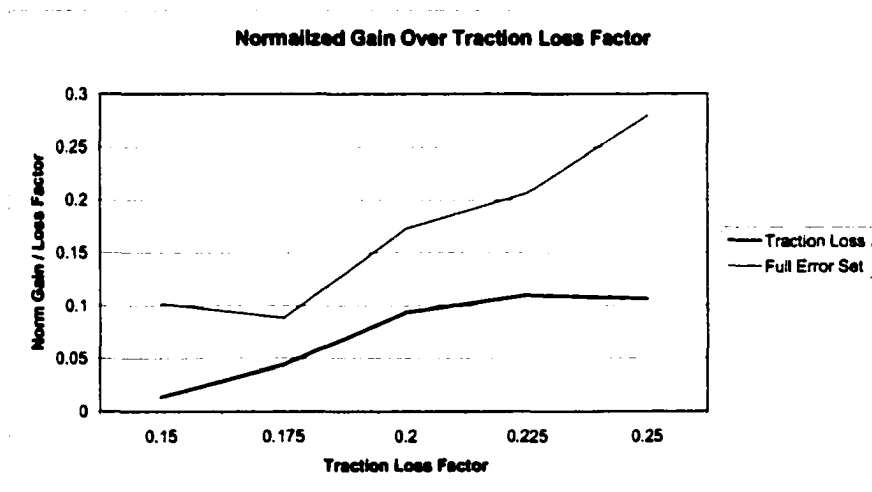


Figure 55. Normalized Gain Over Traction Loss Factor

Examining the *Normalized Gain Over Traction Loss Factor (NGOTLF)* defined as:

$$NGOTLF = \frac{\text{Coverage Improvement}}{TLF \cdot (\text{MapWidth} * \text{MapHeight})}$$

44.

where TLF is the Traction Loss Factor. We see that gains obtained from correction in the presence of the full set of traction losses, sensor errors, and rotation errors produces an increasing function even as the traction loss factor increases. The benefit from applying correction is not constant based on the system and environment's inherent design and construction. There is incremental gain as error induced increases, with respect to the experimental simulator utilized for this work. This makes intuitive sense, as more error being present implies that more error could be detected and hence removed.

We also ran several experimental runs utilizing 3 robots in the environment of 3 objects. There was no detected difference in the correction on the final global map or difference in the performance of the three TDT methods utilized when 3 robots were employed as compared to when only 2 robots were employed. This makes intuitive sense as the benefits of utilizing the MDL paradigm and selecting specific TDT mechanisms are not dependent on the number of mapping agents employed but rather only affects the quality of each agent's local map. The gain seen is a faster completion time, as each agent had to map a smaller portion of the overall environment. The gain is also seen as higher quality maps are combined to form the global map. The quality improvement is obtained as more mapping agents in the field can carry or position more waypoints or markers and thus agents more frequently encounter them for correction of position information and correcting the data collected.

There is an intuitive trade off as one increases the number of mapping agents. Utilizing more agents to map a region of fixed size would naturally result in faster completion time and a more robust overall system. Each agent would map a smaller part of the overall terrain and would thus have to travel shorter distances and induce less error into the local map. This lesser error would require less correction. However, at a certain point, the space traveled by an agent may be so small that they do not encounter any significant number of waypoints, beacons or other correction assisting tools and would then not be able to perform any high quality of correction. It is thus theoretically possible to have the quality of the global map get worse as the number of agents surpasses a given limit. This would be in spite of the greater overlap of neighboring local maps obtained by using more agents. It would be another direction of future research to determine how to compute the optimal number of agents utilized for a mapping mission, or to compute a limit for which the marginal return for added agents is negligible or negative.

The global maps obtained in experimentation were all qualitatively identical. The maps are of course not exactly the same, but to a human observer, there is no significant noticeable difference and a person would likely say the maps are the same. Programming of the mapping agents can include a saving of the *last, best* global map in addition to the current working global map. Thus, if there were a case of mapping agents dropping out and we were left with only a single agent that was no longer receiving any other local map feedback, then this *last, best* map can be secured, along with the last working global map, for reporting at mission end. A *last, best* map is a stored copy of a previously constructed global map which may have a more significant coverage or more agents contributing to it. The global maps stored for this purpose of back-up will be limited by the physical memory limits on the agents themselves.

The coverage metric defined and utilized to measure the *goodness* of the constructed maps is useful in an experimental and particularly simulated environment as we have access to the true world model from which all readings and interactions evolve. In a real deployment, the data used to compute the coverage would not be available and there would be some other need for determining the quality of the constructed map.

5.3.3 Synopsis of Results

Given the goals set out and the results obtained from the experiments and their listed results above, we can make the following statements pertaining to our research.

Proposition 1: The terrain acquisition from mobile robot mapping has been improved by utilizing the error correction schemes employed in conjunction with the use of the MDL map representation paradigm.

Justification: Analysis of results of uncorrected maps as well as corrected maps and the qualitative analysis of the maps produced demonstrates that the coverage and thus the correlation between the produced maps and the environmental model has been improved by the use of correction.

Proposition 2: The map acquisition by mobile robot is more robust by utilizing a group of cooperating agents as opposed to a single agent.

Justification: The proposed mapping paradigm utilizing a cooperative group of mobile robots that does not utilize any central planning or central data storage but rather uses completely distributed agents and a distributed processing scheme, will perform in a similar fashion to other systems which utilizes such distributed computing methods. These systems do not fail unless every single agent fails and these systems can reconfigure themselves to compensate for a defective node in the network. The global maps produced on each mapping agent were identical experimentally and so any single surviving agent could provide the last, best global map from a mapping mission.

Proposition 3: The time-dependent transforms proposed in this research result in improvement in the quality of correction done to the maps constructed with the MDL paradigm.

Justification: The analysis of the experimental results obtained from trying three increasingly more complex mechanisms for time-dependent transforms has shown that improved quality of mapping is obtained with the more sophisticated TDTs.

Proposition 4: The generated grid-type maps are sufficient to allow conversion to precise graph based data representations suited to navigation and path planning.

Justification: The experimental results have shown that the qualitative nature of the maps has improved by using the MDL paradigm in conjunction with sophisticated TDTs. Global coverage map analysis results in very clear definitions of object shapes. While global coverage results would not be available in a deployed system, as the world model would not be known, the resultant maps would still be of similar quality. The maps generated from our algorithms and the MDL paradigm can be used to represent a unique graph-based representation of the mapped environment to facilitate path planning and navigation. Using available transformations from grid-type to graph-type representation [Horst-

96][Thrun-98], one can construct graph-type maps of the environment that should be of no less quality than the grid-type maps from which they originate.

Proposition 5: The algorithms presented and the MDL paradigm complete in finite time and practically consume $O(n^2)$ resources where n is the maximum of the width or height of the map.

Justification: The algorithms designed to perform the mapping operations will terminate once a map has been completed within a certain threshold of coverage of a known area or when no additional unknown spaces can be located within some constant attempt bound. The resources utilized by the algorithms and MDL paradigm depend on the area, the product of the width by the height, of the terrain to be mapped. The size of the unit of measure depends on the resolution required in the final map but is lower bound by the size of the smallest mapping agent. If we are mapping a 2-d projection of a surface, the area covered by the map, at the resolution required, determines the map size and thus the resources required. If we map a 3-d world then the amount of resources jumps to $O(n^3)$. With non-neutrally buoyant mapping agents, the mapping is restricted to discrete levels in such a third dimension and this typically results in the size of the third dimension being much smaller in magnitude than the other two dimensions. If discrete levels can be detected, the algorithms can be designed to stack 2-d maps where a single 2-d map represents each discrete level, rather than employing a fully defined 3-d map which would be very sparse in the third dimension.

6. Conclusion and Future Work

6.1 Contribution of this Work

We proposed a general computational framework for mobile robotic mapping that drew from the benefits of research in related areas and applied those techniques deemed beneficial. We proposed a new paradigm for storing the data collected by the robot agent which offers a great degree of flexibility in its design for on-line correction in the event a systems fault or error in signal data were discovered. We subsequently implemented a simulator to test robot mapping algorithms and implemented a subset of the functionality proposed in the Map Description Language (MDL). We developed several mechanisms used to apply correction factors to the maps stored in the MDL form that apply correction factors to the map more effectively than a uniform correction. Finally, we ran some simulated mapping missions on a variety of simple objects as well as a complex cooperative mapping mission, to test the effect of applying map correction via the historical data recorded in the MDL data stream and to test the performance of the developed TDT mechanisms.

The simulations allowed us to see that correction of data, once an error in position is discovered, does provide a benefit to the accuracy of the generated map. Moreover, it was clear that the benefit gained from utilizing the MDL paradigm increased as the quantity of error sources and error magnitude increased. We also demonstrated that mapping of unknown terrain through a cooperative effort of multiple mapping agents provides benefit in robustness as well as error detection and correction not available in a single robot system. It was clear that the availability of some mechanism by which to detect error in position, either via landmarks or intelligent beacons is needed in unknown environments.

The contributions can be summarized as follows:

1. Most mobile robotic map building work deals with map construction utilizing a single robot and solutions are based on the premise of a single robot.
2. The limited work in cooperative robot map construction in the literature does not allow for the complexities addressed in the single-robot mapping work.

3. None of the map construction research investigated, has dealt with explicit correction of detected error. The single robot mapping research addresses it only in the way new sensor data is integrated into the maps but cannot change that map data once it has been integrated.
4. Our solution provides for a cooperative, multi-robotic mapping solution that allows for the explicit correction of error in maps in an effective way, as experimentally demonstrated.
5. We have introduced a data storage paradigm to assist in map storage, while facilitating correction, with an eye for the complexities of map construction that will come with the new directions mobile robotics is taking. We have developed some schemes for applying map correction once error in the map is detected.

6.2 Future Directions of this Research

There are several areas of interest that were not treated in detail in this work that warrant further investigation by researchers.

The investigation of *completion tests* for determining when the generated map is complete or complete enough is of interest. The completion test utilized for experiment one was the return of the robot to the starting point of its mapping journey around the object. For experiment two, the map was considered complete once the coverage statistic reached a predetermined cut-off level. In relation to this, one must address the space that is *non-mappable* as the result of some free space in the environment not being able to be explored via a mapping agent. This can occur if there is a donut in the environment as we then have an open space with no access to it. It can also occur if the configuration of all of the functional robots is such that none are able to explore the remaining unknown spaces, for example, by the access or space being too small in comparison to the robot sizes. If any of the objects we utilized had been hollow or had an access way that was impassible by any mapping agent, we would not have been able to map their interior. There would thus be a region of the world we would never have been able to map and this could result in a condition where our completion test could never be satisfied. Determining how a set of mapping robots knows when it is finished or at least when the end product is *good enough* is of importance to deploying these solutions into a real world environment.

It would be worth investigating the distribution of the waypoints used as beacons for correction of the dead-reckoning position information. In our experiments, we positioned them somewhat arbitrarily, however it would be of interest to study way of placing a fixed number of such waypoints in an initially unknown terrain. Also of interest would be the number of waypoints that should be deployed over a region of some fixed size with some assumption about the number of obstacles to be encountered. We would want to carry just enough waypoints into a mission and not have to load down with an unnecessarily high number of markers or waypoints that will never be used and take up valuable cargo space or weight. This would be critical information to a space deployed mapping mission to another planet. Along the same line, investigating the number of agents to be used and at which point additional agents provide no benefit or may reduce global map quality would also be of interest.

Another component of the mapping architecture, which was not explored in this work, is the explicit detection and handling of dynamic objects in the environment. We address dynamic objects indirectly by mapping them as stationary objects and then resolving their status in the global map. Lacking repeated observation, a dynamic object fades in the global map as conflicting local maps are combined to form the global map. A way to detect and classify true dynamic features of the environment rather than paint them on the map as stationary objects, would be useful. Such knowledge can provide symbolic information to the end user about the environment not strictly represented on the map itself. Such dynamic objects could be labeled on the symbolic level of the hybrid map as being regions of dynamic objects. If the dynamic object is encountered significantly often enough, we could indicate it may be constantly present in the region, or ignored if it was a transient object to the mapping area such as a squirrel running across a field.

Beyond that, it would be worthwhile to explore the experimental testing into all aspects supported by the MDL system to include a true 3 dimensional world model, including robots of various configurations and including more sensors. NASA, in particular, is working on mobile robotics systems for exploration and labor tasks that can configure themselves to best solve the mission problem. As a result, experiments with real configurable robots and the benefit of the MDL paradigm on maps constructed by such robots would be of interest.

Finally, the utilization of the hybrid grid model for map representation could be tested in conjunction with terrain models tied to the hybrid map to indicate terrain types or hazards. If terrain information can be obtained, agents could utilize this to make further improvements to the TDT utilized for the application of correction factors by applying more correction in those regions where, for example, traction losses are greatest, and less correction where traction is very good. This would improve the robustness of a multi-agent mapping system and also generate a richer map end product.

Extending the hybrid-grid map structure to include feature recognition to facilitate landmark location and referencing would be interesting. We utilized intelligent waypoint beacons as our reference objects for position correction. Being able to utilize features of the environment that stay in place would be beneficial. The key to accurate mapping is primarily tied to maintaining an accurate position reference and the more mechanisms available to an agent during a mapping operation, the more frequently it can correct its position and apply correction factors.

6.3 Final Words

These final thoughts pertain not to the research performed or the results obtained but to the issues of assembling this document for acceptance by the graduate school. Without a doubt, this process is one of the most annoying and painstaking jobs that had to be done. We are almost certain that the poor quality of functions within Microsoft Word 97, in terms of its typesetting capacities and its document handling, are to blame. I can only recommend that anyone attempting to build a dissertation or thesis document utilize LaTeX® or some form of software based on it as it seems to do a much better job at the typesetting, despite its age, than MS Word ever will. Microsoft never did get Word right in this regard and I have little hope it will in the future unless I get in there and fix it from the inside by joining the company. To any students reading this, please heed this warning.

Bibliography

- [Arkin-87] R. Arkin, "Towards cosmopolitan robots: intelligent navigation in extended man-made environments," *Dissertation*, University of Massachusetts, 1987.
- [Arleo-99] A. Arleo, J. Millan, D. Floreano, "Efficient learning of variable-resolution cognitive maps for autonomous indoor navigation," *IEEE Transactions on Robotics and Automation*, v. 15, no. 6, pp. 990-1000, Dec. 1999.
- [Asada-90] M. Asada, "Map building for a mobile robot from sensory data," *IEEE Transactions on Systems, Man, and Cybernetics*, v. 37, n. 6, Nov. / Dec. 1990.
- [Atiya-93] S. Atiya, G. Hager, "Real-time vision-based robot localization," *IEEE Transaction on Robotics and Automation*, v. 9, n. 6, p. 785, Dec. 1993.
- [Ayache-89] N. Ayache, O. D. Faugeras, "Maintaining representations of the Environment of a Mobile Robot," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 6, pp. 804-819, 1989.
- [Balakrishna-95] R. Balakrishna, A. Ghosal, "Modeling of slip for wheeled mobile robots," *IEEE Transactions on Robotics and Automation*, v. 11, n. 1, p. 126, Feb. 1995.
- [Barshan-95] B. Barshan, H. Durrant-Whyte, "Inertial navigation systems for mobile robots," *IEEE Transactions on Robotics and Automation*, v. 11, n. 3, p. 328, Jun. 1995.
- [Basye-89] K. Basye, T. Dean, J. Vitter, "Coping with uncertainty in map learning," *Autonomous Mobile Robots*, ed. S. Iyengar, A. Elfes, v.1.
- [Basye-93] K. J. Basye, "A framework for map construction," *Dissertation*, Brown University, May 1993.
- [Betge-Brezetz-95] S. Betge-Brezetz, P. Hebert, R. Chatila, M. Devy, "Object-based modeling and localization in natural environments," *IEEE International Conference on Robotics and Automation*, p. 2920, 1995.
- [Betge-Brezetz-96] S. Betge-Brezetz, P. Hebert, R. Chatila, M. Devy, "Uncertain map making in natural environments," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1048-1053, 1996.
- [Betke-97] M. Betke, L. Gurvits, "Mobile robot localization using landmarks," *IEEE Transactions on Robotics and Automation*, v. 13, n. 2, p. 251, Apr. 1997
- [Borenstein-91] J. Borenstein, Y. Koren, "Histogramic in-motion mapping for mobile robot obstacle avoidance," *IEEE Transactions on Robotics and Automation*, v.7, no. 4, Aug. 1991.
- [Brooks-85] R. Brooks, "Visual map making for mobile robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, p. 824, 1985.
- [Brooks-86] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, v. RA-2, n. 1, p. 14, Mar. 1986.

- [Braunegg-93] D. Braunegg, "MARVEL: A system that recognizes world locations with stereo vision," *IEEE Transactions on Robotics and automation*, v. 9, n. 3, p. 303, Jun. 1993
- [Bulata-96] H. Bulata, M. Devy, "Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot," *Proceedings of the IEEE International Conference on Robots and Systems*, pp.1054-1060, 1996.
- [Cela-92] A. Cela, Y. Haman, "Optimal motion planning of a multiple-robot system based on decomposition coordination," *IEEE Transactions on Robotics and Automation*, v. 8, n. 5, p. 585, Oct. 1992.
- [Chatila-85] R. Chatila, J-P Laumond, "Position referencing and consistent world modeling for mobile robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 138-145, Mar. 1985.
- [Cox-94] I. Cox, J. Leonard, "Modeling a dynamic environment using a bayesian multiple hypothesis approach," *Artificial Intelligence*, n. 66, pp. 311-344, 1994.
- [Cozman-95] F. Cozman, E. Krotkov, "Robot localization using a computer vision sextant," *IEEE International Conference on Robotics and Automation*, p. 106, 1995.
- [Davison-01] A. Davidson, N. Kita, "Sequential localization and map-building in computer vision and robotics," *SMILE 2000*, pp. 218-234, 2001
- [Deveza-94] R. Deveza, D. Thiel, A. Russell, A. Mackay-Sim, "Odor sensing for robot guidance," *International Journal of Robotics Research*, v. 13, n. 3, p. 232, June 1994.
- [Elfes-86] A. Elfes, "A distributed control architecture for an autonomous mobile robot," *Artificial Intelligence*, Vol. 1, No. 2, 1986
- [Elfes-86b] A. Elfes, "A sonar-based mapping and navigation system," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1151-1156, 1986.
- [Elfes-86c] A. Elfes, "A distributed control architecture for an autonomous mobile robot," *Artificial Intelligence*, v. 1, n. 2, p. 135, 1986.
- [Elfes-87] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, pp. 249-265, 1987.
- [Elfes-89] A. Elfes, "Occupancy grids: a probabilistic framework for robot perception and navigation," *Dissertation*, Carnegie-Mellon University, May 1989.
- [Elfes-90] A. Elfes, "Occupancy Grids: A stochastic spatial representation for active robot perception," *Proceedings of the Sixth Conference on Uncertainty in AI*, July 1990.
- [Fok-92] K. Fok, M. Kabuka, "A flexible multiple mobile robot system," *IEEE Transactions on Robotics and Automation*, v. 8, n. 5, Oct. 1992.
- [Freedman-85] P. Freedman, G. Carayannis, D. Gauthier, A. Malowany, "A session layer for a distributed robotics environment," *Proceedings of the IEEE COMPINT Conference*, p. 459, Sept. 1985.

- [Fryer-95] J. Fryer, G. McKee, "Exploring cooperative intelligence in a networked robotic system," *ISCA 4th Golden West Conference on Intelligent Computer Systems*, Jun. 1995.
- [Fujimura-94] K. Fujimura, "Motion planning amid transient obstacles," *The International Journal of Robotics Research*, v. 13, n. 5, p. 395, Oct. 1994.
- [Gaglianello-86] R. Gaglianello, H. Katseff, "A distributed computing environment for robotics," *IEEE*, p. 1890, 1986.
- [Gat-94] E. Gat, R. Desai, R. Ivlev, J. Loch, D. Miller, "Behavior control for robotic exploration of planetary surfaces," *IEEE Transactions on Robotics and Automation*, v. 10, n. 4, p. 490, Aug. 1994.
- [Gauthier-87] D. Gauthier, P. Freedman, G. Carayannis, A. Malowany, "Interprocess communication for distributed robotics," *IEEE Journal of Robotics and Automation*, v. RA-3, n. 6, p. 493, Dec. 1987.
- [Harmon-87] S. Harmon, "The ground surveillance robot (GSR): An autonomous vehicle designed to transit unknown terrain," *IEEE Journal of Robotics and Automation*, v. RA-3, n. 3, p. 266, Jun. 1987.
- [Horst-96] J. Horst, "Maintaining Multi-level Planar Maps in Intelligent Systems," *IEEE International Conference on Robots and Systems, IROS96*, pp. 1061-1066, 1996.
- [Iyengar-94] S. Iyengar, D. Jayasimha, D. Nadig, "A versatile architecture for the distributed sensor integration problem," *IEEE Transactions on Computers*, v. 43, n. 2, p.175, Feb. 1994.
- [Khatib-86] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, v. 5, n. 1, p. 90, 1986.
- [Kabuka-87] M. Kabuka, A. Arenas, "Position verification of a mobile robot using standard pattern," *IEEE Journal of Robotics and Automation*, v. RA-3, n. 6, p. 505, Dec. 1987.
- [Kosaka-93] A. Kosaka, M. Meng, A. Kak, "Vision-guided mobile robot navigation using retroactive updating of position uncertainty," *Proceedings of the 1993 IEEE international Conference on Robotics and Automation*, v. 2, p. 1, 1993.
- [Krotkov-89] E. Krotkov, "Mobile robot localization using a single image," *Proceedings of the IEEE International Conference on Robotics and Automation*, p.978, 1989.
- [Krotkov-95] E. Krotkov, M. Hebert, "Mapping and positioning for a prototype lunar rover," *Transactions of the IEEE International Conference on Robotics and Automation*, p. 2913, 1995.
- [Leonard-92] J. J. Leonard, H. Durrant-Whyte, I. Cox, "Dynamic map building for an autonomous mobile robot," *International Journal of Robotics Research*, v. 11, no. 4, Aug. 1992.
- [Lu-95] F. Lu, E. Milios, "Optimal global pose estimation for consistent sensor data registration," *IEEE International Conference on Robotics and Automation*, p. 93, 1995.

- [Lu-97] F. Lu, E. Miliotis, "Robot pose estimation in unknown environments by matching 2D range scans," *Journal of Intelligent and Robotic Systems*, n. 18, pp. 249-275, 1997.
- [Lyons-93] D. Lyons, "Representing and analyzing action plans as networks of concurrent processes," *IEEE Transactions on Robotics and Automation*, v. 9, n. 3, p. 241, Jun. 1993.
- [Melliari-Smith-90] P. Melliari-Smith, L. Moser, V. Agrawala, "Broadcast protocols for distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, v. 1, n. 1, p. 17, Jan. 1990.
- [Mitchell-84] J. Mitchell, D. Keirse, "Planning strategic paths through variable terrain data," *SPIE Vol. 485 - Applications of Artificial Intelligence*, pp. 172-129, 1984.
- [Moravec-85] H. P. Moravec, A. Elfes, "High resolution maps from wide angle sonar," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 116-121, 1985.
- [Nitzan-85] D. Nitzan, "Development of intelligent robots: achievements and issues," *IEEE Journal of Robotics and Automation*, v. RA-1, n. 1, p. 3, Mar. 1985.
- [Noreils-93] F. Noreils, "Toward a robot architecture integrating cooperation between mobile robots: application to indoor environment," *International Journal of Robotics Research*, v. 12, n. 1, p. 79, Feb. 1993.
- [Noreils-95] F. Noreils, R. Chatila, "Plan execution monitoring and control architecture for mobile robots," *IEEE Transactions on Robotics and Automation*, v. 11, n. 2, p. 255, Apr. 1995.
- [Oriolo-95] G. Oriolo, M. Vendittelli, G. Ulivi, "Online map building and navigation for autonomous mobile robots," *IEEE International Conference on Robotics and Automation*, p. 2900, 1995.
- [Papageorgiou-94] M. Papageorgiou, T. Bauschert, "Stochastic optimal control of moving vehicles in a dynamic environment," *The International Journal of Robotics Research*, v. 13, n. 4, pp. 343-354, Aug. 1994.
- [Prassler-95] E. Prassler, E. Miliotis, "Position estimation using equidistance lines," *IEEE International Conference on Robotics and Automation*, p. 85, 1995.
- [Russell-93] R. A. Russell, "Mobile robot guidance using a short lived heat trail," *Robotica*, v. 11, pp. 427-431, 1993.
- [Santos-96] V. Santos, J. Goncalves, F. Vaz, "Local perception maps for autonomous robot navigation," *Proceedings of IROS 96*, p. 821, 1996.
- [Schenker-00] P. S. Schenker et al, "Reconfigurable robots for all terrain exploration," *Proceedings of SPIE*, v. 4196, *Sensor Fusion and Decentralized Control in Robotic Systems III*, Nov. 2000.

- [Schenker-01] P. S. Schenker, T. L. Huntsberger, P. Pirjanian, E. T. Baumgartner, "Planetary rover developments supporting Mars science, sample return and future human-robotic colonization," Invited paper to appear in *Proceedings of the 10th International Conference on Advanced Robotics*, Budapest, Hungary, Aug. 2001.
- [Shatkay-97] H. Shatkay, L. P. Kaelbling, "Learning topological maps with weak local odometric information," *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pp.920-927, Aug. 1997.
- [Shen-85] H. Shen, G. Signarowski, "A knowledge representation for roving robots," *IEEE Second Conference on Artificial Intelligence Applications*, pp. 621-682, Dec. 1985.
- [Singh-93] L. Singh, K. Fujimura, "Map making by cooperating mobile robots," *IEEE International Conference on Robotics and Automation*, v. 2, p. 254, 1993.
- [Smailus-98] T. Smailus, S. Iyengar, "Information management and fusion for efficient robot utilization in a dynamic manufacturing environment," *International Conference on Information Technology Integration for Manufacturing*, Dec. 1998.
- [Smailus-01] T. Smailus, S. Iyengar, "Map storage technique for heterogeneous and distributed robotic mapping," submitted to the *Intelligent Automation and Soft Computing*, 2001.
- [Smith-89] R. Smith, M. Self, P. Cheeseman, "A stochastic map for uncertain spatial relationships," *Autonomous Mobile Robots*, ed. S. Iyengar, A. Elfes, v. 1, p. 323.
- [Sugihara-88] K. Sugihara, "Some location problems for robot navigation using a single camera," *Computer Vision, Graphics, and Image Processing*, v. 42, pp.112-129, 1988.
- [Taluri-92] R. Talluri, J. K. Aggarwal, "Position estimation for an autonomous mobile robot in an outdoor environment," *IEEE Transactions on Robotics and Automation*, v. 8, n. 5, p. 573, Oct. 1992.
- [Thrun-98] S. Thrun, "Learning metric-topology maps for indoor mobile robot navigation," *Artificial Intelligence*, v. 99, pp. 21-71, 1998.
- [Thrun-98b] S. Thrun, W. Burgard, D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Machine Learning*, v. 31, pp. 29-53, 1998.
- [Tsubouchi-96] T. Tsubouchi, "Nowadays trends in map generation for mobile robots," *Proceedings of the IEEE International Conference on Robots and Systems*, pp. 828-833, 1996.
- [Turchan-85] M.P. Turchan, A.K.C. Wong, "Low level learning for a mobile robot: environment model acquisition," *IEEE Second Conference on Artificial Intelligence*, p. 156, Dec. 1985.
- [Weigl-93] M. Weigl, B. Siemiatkowska, K. A. Sikorski, A. Borkowski, "Grid-based mapping for autonomous mobile robot," *Robotics and Autonomous Systems*, pp. 13-21, 1993.
- [Wilcox-96] B. Wilcox, "Nanorovers for planetary exploration," *AIAA Robotics Technology Forum*, Madison, WI, pp. 11-1 – 11-6, Aug. 1996.

Vita

Thomas Oliver Smailus was born in Würzburg, Bavaria, in the Federal Republic of Germany on December 17th, 1967. He attended school in both Germany, at Mannheim Elementary, Middle and High Schools, and various locations in the United States of America (Ft. Carson, Colorado Springs, Colorado; Ft. Jackson, Columbia, South Carolina; and Ft. Polk, Leesville, Louisiana) as the result of being in a military family. He received a bachelor of science degree in computer engineering from Louisiana State University in the spring of 1990, a master of science degree in electrical engineering from Louisiana State University in the spring of 1992, and a doctor of philosophy degree in computer science from Louisiana State University in the summer of 2001.

Mr. Smailus has been writing software and designing hardware systems since 1983. He has been active as a systems and network administrator at the College of Engineering at Louisiana State University where he was also active as a researcher in the areas of remote sensing, traffic engineering, database systems and internet applications programming. He has also worked at Asea Brown Boveri in Mannheim, Germany, as an intern. Mr. Smailus has been a licensed Professional Engineer in Electrical Engineering in the state of Louisiana since 1997. His research interests include digital systems design, software design for network and internet application, embedded systems, computer architecture, operating systems, distributed processing, hardware interfacing and mobile robotics.


DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Thomas Oliver Smailus

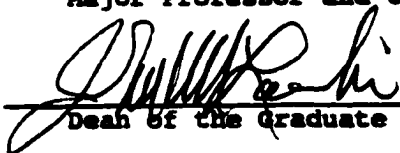
Major Field: Computer Science

Title of Dissertation: Mapping by Cooperative Mobile Robots

Approved:

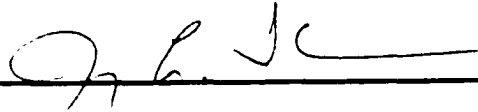



Major Professor and Chairman

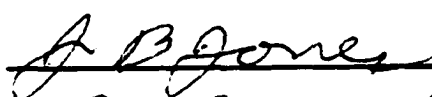


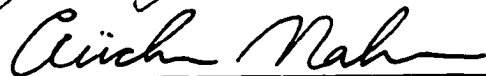
Dean of the Graduate School


EXAMINING COMMITTEE:











Date of Examination:

June 6, 2001