3-2004

Concurrent Cognitive Mapping and Localization Using Expectation Maximization

Kennard R. Laviers

CONCURRENT COGNITIVE MAPPING AND LOCALIZATION USING EXPECTATION MAXIMIZATION

THESIS

Kennard R. Laviers, First Lieutenant, USAF

AFIT/GCS/ENG/04-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/04-10

Concurrent Cognitive Mapping and Localization using Expectation Maximization

THESIS

Presented to the Faculty of the
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

Kennard R. Laviers, B.S.
First Lieutenant, USAF

March, 2004

AFIT/GCS/ENG/04-10

Concurrent Cognitive Mapping and Localization using Expectation Maximization

Kennard R. Laviers, B.S.
First Lieutenant, USAF

Approved:

/signed/	18 Feb 2004
Doctor Gilbert L. Peterson Thesis Advisor	Date
/signed/	18 Feb 2004
Doctor Henry B. Potoczny Committee Member	Date
/signed/	18 Feb 2004
Doctor John F. Raquet Committee Member	Date
/signed/	18 Feb 2004
Major Timothy S. Webb Committee Member	Date

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abstract

Robot mapping remains one of the most challenging problems in robot programming. Most successful methods use some form of occupancy grid for representing a mapped region. An occupancy grid is a two dimensional array in which the array cells represents (x, y) coordinates of a cartesian map. This approach becomes problematic in mapping large environments as the map quickly becomes too large for processing and storage.

Rather than storing the map as an occupancy grid, our robot (equipped with ultrasonic sonars) views the world as a series of connected spaces. These spaces are initially mapped as an occupancy grid in a room-by-room fashion using a modified version of the Histogram In Motion Mapping (HIMM) algorithm extended in this thesis. As the robot leaves a space, denoted by passing through a doorway, it converts the grid to a polygonal representation using a novel edge detection technique. Then, it stores the polygonal representation as rooms and hallways in a set of Absolute Space Representations (ASRs) representing the space connections. Using this representation makes navigation and localization easier for the robot to process. The system also performs localization on the simplified cognitive version of the map using an iterative method of estimating the maximum likelihood of the robot's correct position. This is accomplished using the Expectation Maximization algorithm. Treating vector directions from the polygonal map as a Gaussian distribution, the Expectation Maximization algorithm is applied, for the first time, to find the most probable correct pose while using a cognitive mapping approach.

Concurrent Cognitive Mapping and Localization using Expectation Maximization

I. Introduction

Learning a map of its environment presents a complex problem for robots. The problem begins with getting the robot to move in such a fashion that it explores the entire environment, insuring a complete map is constructed. While the robot explores and records real time sensor data, it uses that data (range and motor data) to form a map of the explored region. These maps allow the robot to develop a sense of a place and localize themselves in the world. Memory space limitations (memory is finite in any robot) require creative solutions for the representation of the map itself when mapping large environments. These maps can take on a number of internal representations from a grid to a general graph of interconnected spaces; our method combines the best of both approaches, allowing for greater efficiency performing planning and localization.

An associated problem to learning the map includes localization of the robot. Accurate localization is a prerequisite for building a good map and having an accurate map is essential for good localization. Simultaneous Localization And Mapping (SLAM) is a critical underlying factor for any successful mobile robot navigation.

1.1 Overview

The process of learning a map breaks into three main categories; mapping, localization and exploration. Each of these categories function independently but inevitably depend on each other. A robot maps a region by creating a representation of the area it has explored. The robot can perform exploration without knowledge

of where it has been, but if so, is no more than a random wandering routine just as prone to revisiting an already explored region than not. Localization without consideration of observation data, called dead reckoning, is not very accurate as the tires tend to slip and rotation sensors are not perfectly accurate. For these reasons, it is important to use observation data along with dead reckoning data to gain a more accurate estimation of the robot's location. Thus, there is a mutual dependence between mapping, localizing and exploring. Each of these categories has its own associated challenges as explained in the following sections, starting with mapping.

1.1.1 Mapping. The representation of the map is a key factor of robot mapping. Implementations of robot mapping represent map data as either an occupancy grid [26][4] or a topological network [17][16][13]. An occupancy grid is a matrix of cells where cells containing a value greater than 0.0 indicates a belief that an obstacle resides in space represented by the the cell. Using a matrix in this fashion requires the reservation of a large amount of memory for representing the entire maximum map size. A topological map is one in which objects are stored with respect to each other based on a *robot centric* point of view. This representation makes it easy to use memory only as needed, generating more efficient maps, but introduces implementation difficulties and cannot easily convert to human-understandable maps.

Generally, the robot is equipped with multiple sensors, and as sensor data accumulates, the representation of the map is developed from the fusion of that data. This process is usually referred to as *sensor fusion* [26].

Occupancy grids can be either discrete [4] or probabilistic [26] [27]. Discrete grids produce maps by incrementing and decrementing cell values. Probabilistic methods calculate the probability of a cell being occupied or empty with each update using Bayes probability theorem [26]. Another technique incorporates the Dempster Shafer-Belief theorem [27] to find the belief of the occupancy status of a cell by representing each cell's occupation with Shafer's theory of belief and combining the

multiple sensors with Dempster's combination theory. Probabilistic approaches tend to generate more accurate and detailed maps than of the discrete methods, but are slower due to the increase in the number of calculations.

Topological mapping, a more recent mapping approach, maps the world much like a graph with nodes and edges as paths between locations. We can separate the map into rooms or spaces and assign them as nodes in a graph connected by, in the case of rooms; the exits, and in the case of spaces; the location that separates the spaces. The issue still remains of how to represent the rooms or spaces. In this thesis, the robot creates the room and spaces representation as a set of line segments or polylines maintaining geometric spacial elements. This data representation significantly reduces the size of the data structure and also lends itself to faster path planning, a more human centric representation and the ability to localize without using local landmarks. However, generating the polygonal representation with sonars remains difficult since the noise generated by the sonars significantly complicates the process.

We overcome the issue of sonar noise by using one of the sensor fusion techniques (discussed above) with a grid representation first, then converting the grid representation to a topological map. This process starts by creating a histogram map or occupancy grid from the sonar input/data. Using a new technique of filtering the data in the histogram (developed in this thesis), which reduces the histogram wall thickness from thickness of as much as 10 cells to one, the data is cleaned and then converted to a list of vertices and filtered using the Douglas-Peucker line reduction algorithm [9]. The remaining set of lines represents the original map of the room. These lines, along with other topological information, are stored in an Absolute Space Representation (ASR) data structure and used later to form a complete map. This process repeats for each room or space visited.

1.1.2 Localization. As the robot maps its environment, errors accumulate within the robots dead reckoning system causing incorrect pose data and consequently the map to skew (Fig. 1.1). The coordinates at which the robot believes itself to be located is the robots pose. Localization is the process of determining where in the environment the robot is located, and the process of correcting errors in the robot's pose.

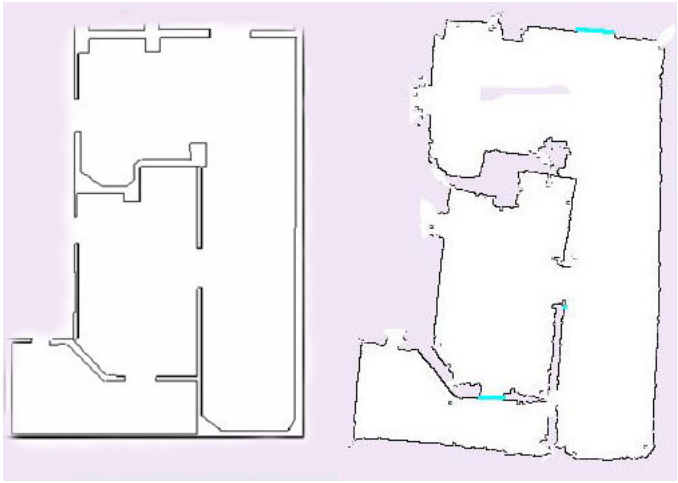


Figure 1.1 This map shows skewing caused by dead reckoning error. The room in the middle is overlapping with the large room to right.

Localization consists of two broad categories: local and global. Local techniques compensate for odometer errors and require that the initial location of the robot is approximately known. Localization in this fashion progresses iteratively, i.e. after some fixed amount of distance traveled, the robot analyzes motor and sensor data to determine an adjustment of the robot's pose. Local techniques are weak in that the robot usually cannot recover true pose information if the local techniques lose track of the robot's immediate position. Global techniques are stronger as they localize the robot without any prior knowledge about its initial position. They can also handle the *kidnapped robot problem*, wherein a robot is kidnapped and carried to some unknown location. This technique, if no prior map is provided, waits until the robot completes mapping then determines the most likely map based on all the

data. If a map is provided, the robot is able to localize its whereabouts at any given time. Global localization techniques perform better than local ones and deal with situations in which the robot is likely to experience serious positioning errors [32]. Our system performs localization on a simplified cognitive version of the map as an iterative method of estimating the maximum likelihood of the correct position of the robot for each pose. This is accomplished with the use of the Expectation Maximization algorithm [8].

1.1.3 Simultaneous Localization and Mapping. A mapping technique which performs both mapping and localization concurrently generates a more accurate map. When operating together, this process is referred to as Simultaneous Localization and Mapping (SLAM) or Concurrent Localization and Mapping (CLM). SLAM is done either iteratively or once at the end. If the iterative approach is used, then it localizes while the robot travels and maps. If the batch technique is used, then mapping and motor data is stored while the robot travels. After stopping, the robot compiles the data into a complete map using all available data. The later method has proved the most successful in recent work using the Expectation Maximization algorithm [36].

Expectation Maximization SLAM [36] (explained in greater detail in Section 2.3), perhaps the most successful SLAM algorithm available, provides the inspiration for the localization method performed in this thesis. Expectation Maximization is a general algorithm used to iteratively calculate the maximum likelihood function of some probability distribution function. The term "Expectation Maximization" algorithm was coined by Dempster et al. [8]. The maximum likelihood function (taken from probability theory) calculates the maximized expected value of some probability distributionFor mapping, this probability distribution generally includes obstacles and robot locations in a world format.

1.1.4 Exploration and Planning. Another key area of research in robotic mapping is exploration. Does the robot wander randomly, or does it actively explore areas that it has not yet experienced? Complicating the process, the robot should move from explored to unexplored areas in such a way that it minimizes distance traveled while maximizing new mapped areas. To actively explore the region, the robot must know how to move from its current location to another location (Fig. 1.2). This requires that the robot have some criterion for selecting what is unexplored and which unexplored region to travel to. This requires some path planning on part of the robot. A natural extension that we propose to our technique is an exploration feature that plans using simplified polyline data. More on the topic is discussed in Section 2.1.2.

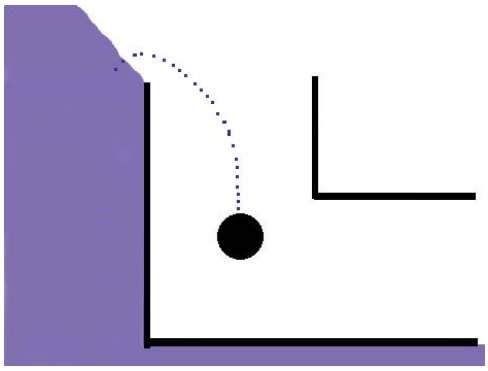


Figure 1.2 The robot travels in the heading that is most unexplored. This is the shaded area on the left and bottom of the figure.

1.2 Research Goals and Objectives

Almost every robot system requires some sort of internal representation of the world that the robot occupies. In today's military, because of land mines and other deadly traps, applications needing robots to scout and learn environments before humans enter them are apparent. The goals of this research are to:

- map a region quickly,
- map an area as accurately as possible,

- provide mapping for large indoor areas,
- and represent the map using little memory.

The objectives used to accomplish these goals are:

1. Modify the histogram mapping algorithm to provide a map suitable to convert to polylines.
2. Develop a new edge detection algorithm capable of quick and accurate edge detection with the histogram map.
3. Create a method to represent histogram map data as polylines.
4. Develop a topological representation of the polylines, separating spaces or rooms of the map and representing them as nodes in a graph.
5. Implement and test a new algorithm using Expectation Maximization to perform mapping and localization corrections on the simplified topological map.

1.3 Approach

The system is developed using a top down approach. In the following section, a layered abstraction is shown of the complete system (Fig. 1.3). At the top is the lowest level or the least abstracted level, and at the bottom is the most abstracted level. That is what is meant by *top down*; modules are designed and implemented starting at the lowest level of abstraction. However, sometimes as new components are added, changes are needed to the previous components. As these components are modified with new integrations they tend to *evolve*. This process of adding and modifying is repeated until the system is complete.

1.4 Software Design Process

Development of a software system requires the use of a structured top down design approach starting with a problem statement and ending with an implemented

system. The overall system is shown in (Fig. 1.3). At the top of the figure some input sensor is used, whether it is sonar, stereo camera, laser(not shown), or some other sensor device. If a camera is used, data from the camera goes first to an image processing subsystem then to the histogram mapping algorithm. Using sonars, the data is sent directly to the mapping algorithm. The next layer under the mapping itself is where the interesting work is done. In this layer, the system converts the raw map to polylines, topologically represented. Also, localization occurs simultaneously with the topological map creation at this stage. If path planning is to be used it would be placed at this level as well.

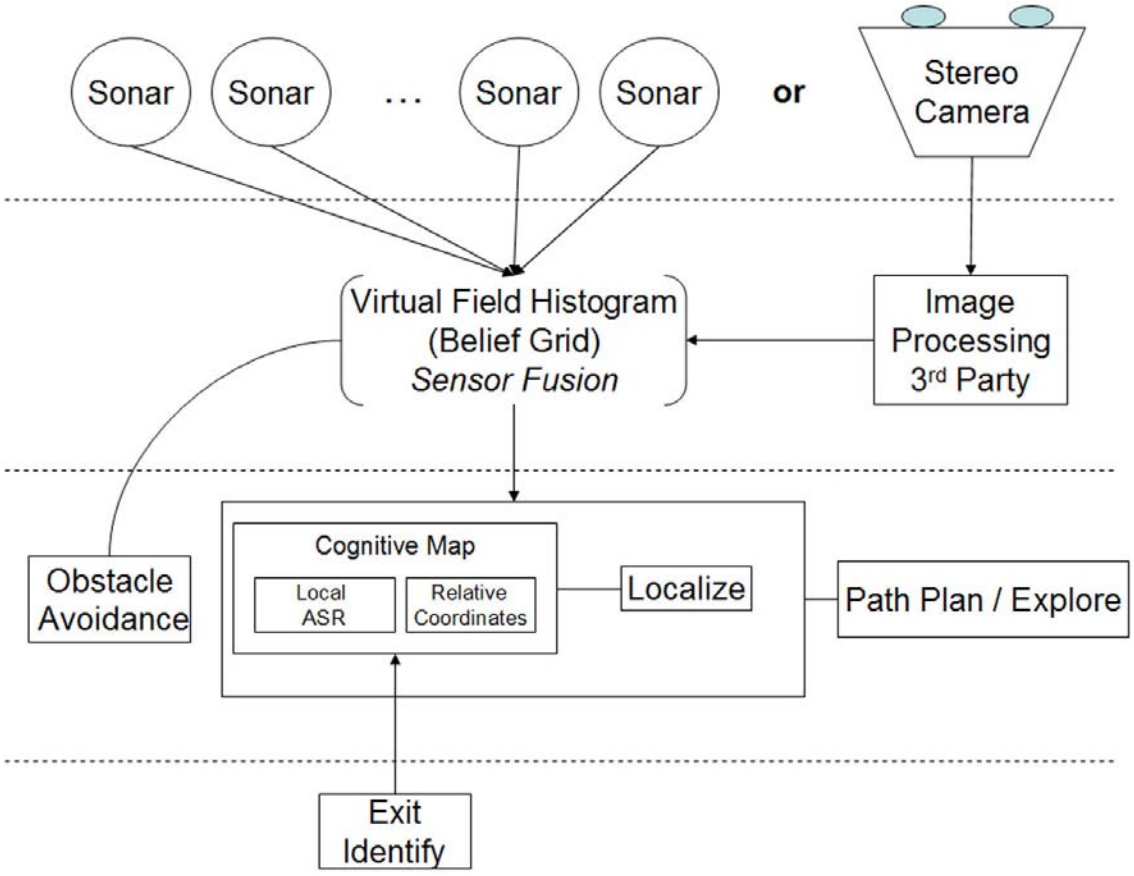


Figure 1.3 System dataflow

To build this system the following steps are repeated for each module of the system [22]:

Step 1 Define/analyze domain requirements for each module.

Step 2 Develop a general solution and operational design specification using algebraic or symbolic notation.

Step 3 Refine the solution design to a low-level design incorporating additional data structures and operations required to create an algorithmic design template.

Step 4 Map the algorithm template to a low level programming language.

1.5 Assumptions

Robotic mapping has many elements tied to computer science and mathematics. To reduce the scope of discussion, the following assumptions of the readers academic background are made:

1. General understanding of trigonometry and geometry
2. Basic calculus
3. Comfortable with probability and set theory
4. General computer science background

Image manipulation requires an understanding of trigonometry and geometry. Moreover, sensor fusion and mapping in general is steeped in calculus, probability and set theory. This thesis is written assuming that the reader is familiar with these types of math in addition to elementary computer science and robots in general.

1.6 Risks

The Pioneer robot in which the mapping system is built has limited memory and processor capabilities. Because the mapping software requires very close to real time processing, this is the greatest risk. Moreover, the robot is controlled with the ARIA robot architecture and compatibility issues with other standard library templates is also a key risk.

1.7 Thesis Outline

In Chapter II, background information related to robot mapping is discussed along with some key areas of research that are building blocks in this research. In Section 2.2, obstacle mapping is explained in great detail with an emphasis on Borenstein and Koren's technique [4] and how we extend it in our work. In Section 2.4, some cognitive representations of the robot maps are discussed again focusing on the use of ASRs for topological representation. These building blocks are pieced together (along with some new techniques) into a design and implementation in Chapter III. The implementation is tested and results are analyzed of simulated and real world mapping in Chapter IV. Finally, future work and extensions are discussed in Chapter V.

II. Background and Related Work

The goal of robot mapping is to build a useful map and provide a solid method for the robot to identify and avoid obstacles in its path, and if revisiting a location, to recognize that fact. While the robot explores the environment, it records data readings from sensors and stores that data for this purpose. In the following sections we discuss the two main approaches of maintaining and representing this information. A survey of several robot exploration techniques is covered in Section 2.1. In Section 2.2, we explore obstacle mapping and how it relates to this work. Finally, Section 2.4 details cognitive mapping used with polylines and space/room map fragmentation.

2.1 Exploration

Many researchers have studied exploration strategies along with their research of localization and mapping[21][11][20]. Exploration is a pivotal part of the learning process; if we don't explore, how do we learn anything new? Benjamin Kuipers and Yung-Tai Byun[21] use a hill climbing strategy to perform exploration.

2.1.1 Kuipers and Byun Exploration. An algorithm proposed by Benjamin Kuipers and Yung-Tai Byun provides a good method of exploration. They refer to a *place* as a zero-dimensional local maximum, and a *path* followed during exploration is defined by some distinctiveness criterion that is sufficient enough to specify a one-dimensional set of points. The robot follows the midline of a corridor, or walks along the edge of a large space, but does not venture into the interior of a large space. This is because the points have no qualitatively distinctive characteristics, at least to its limited-range sensors. Paths traveled connecting two distinctive places are defined in terms of Local Control Strategies (LCS). A LCS is a strategy for what to do based on the robots current location. Once a place has been identified, the robot selects an appropriate local control strategy for moving into an open direction (that is, a direction that has no obstruction). While following a path chosen, the

robot continues analyzing its input for signs of new distinctive features. Once the next distinctive place is identified, the path connecting the two places is defined by the LCS [21]. This method is deterministic in nature but offers a good guide to structuring a non-deterministic method of accomplishing the same task [21].

2.1.2 Next Best View. When using a more accurate range device such as a laser which only requires one scan, the robots exploration is viewed in terms of views needed for scans. After performing a scan, the robot determines the Next Best View (NBV) which will provide just enough overlap with the current scan as to allow the robot to merge the two scans together properly and update the robots dead reckoning system based on the merged data. Hector Gonzanos and John-Claude Latombe make use of this strategy to perform this type of exploration [11].

Gonzanos and Latombe [11] define a *safe region* as the largest regions that are safe to travel in given sensor readings obtained to that point, while providing the minimum amount of overlap required to perform alignment. Exploration and alignment are accomplished simultaneously, building the map iteratively, merging scans from each *safe region* until the map is complete.

This technique works well but does offer some limitations. As their system only uses a laser which scans only a cross section, obstacles can be missed in the scan. Additionally, errors with the polyline extraction can result in a completely unusable map which will produce incorrect exploration.

2.2 Occupancy Grid

Before beginning our discussion of mapping, it is appropriate to define occupancy grids and how range data returns a positive reading for a specific cell within the occupancy grid. One can think of the proposed map as an image to be created in a two dimensional matrix denoted C. The first step is selecting a value $\wp$ that is a scaler used to determine cell size, and making a determination of the world size to represent

C_{mn} where m is the max distance divided by $\wp$ expected in the x direction. n is the max distance expected to travel in the y direction divided by $\wp$. All calculations are based on a standard cartesian system. It is important to understand that information from the sonars is polar, that is, we know how far away it sees an object and at what angle. The robot has an orientation $h : h = \{-180, -179, -178, ..., 178, 179, 180\}$ relative to the initial orientation of the robot from when the mapping process began. Turns to the left decrease h and turns to the right increase h. Each sonar has an angle $sh_i : i = \{1, 2, 3, ...number\ of\ sonars\}$ relative to the robot. The global angle of each sonar is then $h_i = h + sh_i$. So a cartesian point for sensor i, (x_i, y_i) is calculated by:

$$x_i = \frac{r_i \times \cos(h_i)}{\wp} \tag{2.1}$$

$$y_i = \frac{r_i \times \sin(h_i)}{\wp} \tag{2.2}$$

Cells are updated for each reading returned by the sonars using an update function which is dependent upon the specific mapping algorithm used. Update functions vary between the three main available methods of mapping which are: probabilistic [26], belief [27] and discrete [4]. Generally, the most accurate is the belief method (using the Dempster-Shafer belief theorem) followed by the probabilistic approach. Both probabilistic mapping methods are slower, requiring more computation than discrete mapping techniques. In Section 2.2.1 the first probabilistic mapping technique known as Bayes mapping [26] is discussed, followed by the mapping technique providing a major enhancement known as the Dempster-Shafer method. Finally, in Section 2.2.3, a fast discrete mapping technique which approximates probability values calculated in [26][27], known as Histogram in Motion Mapping (HIMM) [4], is explained in great detail. HIMM, designed for speed, is the fastest and simplest of the algorithms and is the method used and extended in this thesis.

2.2.1 Bayes Mapping. One of the most straightforward methods of representing a map has been the occupancy grid [26]. In this method, the world is represented as a two dimensional array of probability information about the occupancy of a grid cell. Using information from the robot's range sensors and pose, an array cell is updated with new probability information after each sensing action [26]. The Bayesian method sees each sonars data as a probability distribution within a cone of error (Fig. 2.1). The angle of error for the sonar is β. Area I in the cone is the where an obstacle is thought to be. Because a sonar cannot sense through an object, area II is assumed unoccupied.

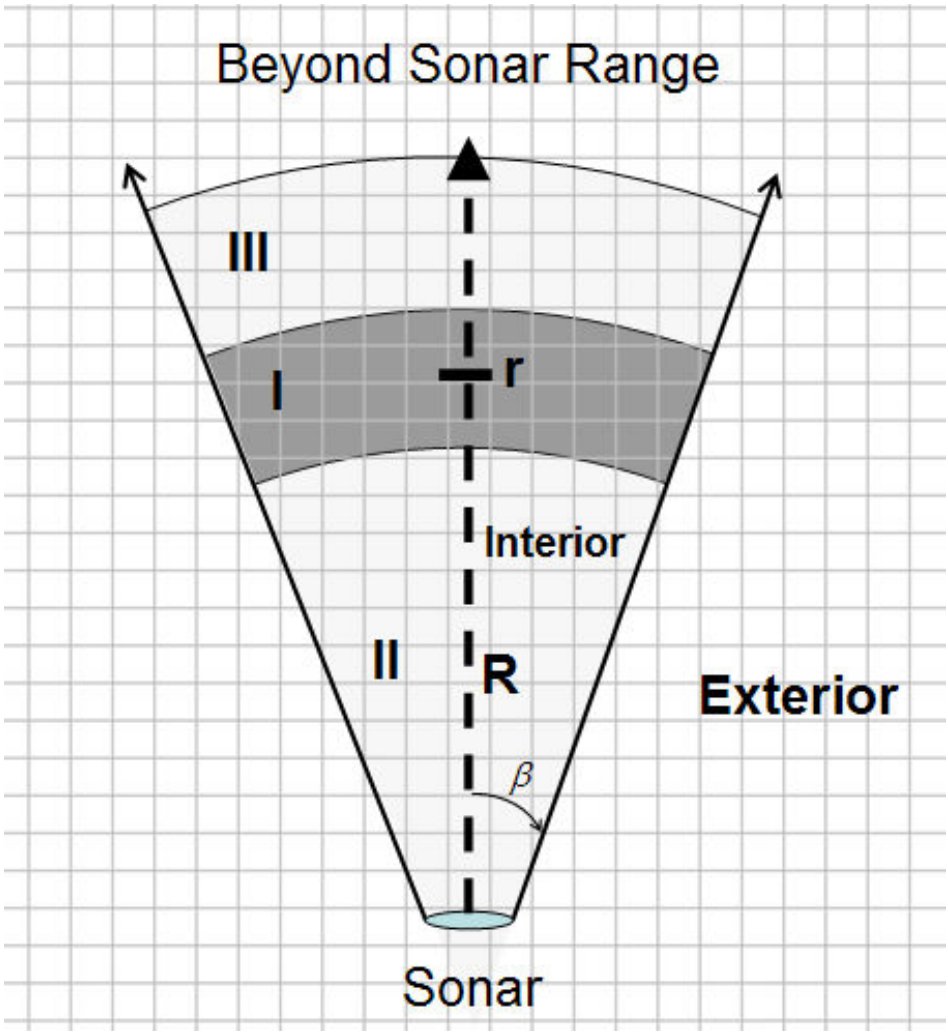


Figure 2.1 Probabilistic Sonar Cone

The Bayes formula ($\neg$ implies negation):

$$\frac{\Pr(o|A,B)}{\Pr(\neg o|A,B)} = \frac{\Pr(A|o,B)}{\Pr(A|\neg o,B)} \times \frac{\Pr(o|B)}{\Pr(\neg o|B)} \tag{2.3}$$

provides a combination of independent data A and B as single data item, $\Pr(o|A,B)$. o is the observation, A is the current data item and B the prior probability. As the sensors create readings independently of each other, each sensor is considered an independent data source. Moravec [26] uses this as a basis to derive the map combining formula:

$$\frac{\Pr(o|A,B)}{\Pr(\neg o|A,B)} = \frac{\Pr(o|B)}{\Pr(\neg o|B)} \times \frac{\Pr(o|A)}{\Pr(\neg o|A)} \times \frac{\Pr(\neg o)}{\Pr(o)}. \tag{2.4}$$

Let $\Pr(r)$ prior to a reading be the probability that the next reading results in measurement r. Denote each possible range R_i (for example, if the sonars have a max range of 10m and our cell size is 1m each then $i = \{0, 1, 2, ..., 10\}$). To find the probability for each R use [26]:

$$\Pr(R_i) = S(R_i) \times \left(1 - \sum_{j=0}^{i} \Pr(R_j)\right) \tag{2.5}$$

where $S(R_i)$ is the sum of the products at R_i. The quantity e_{miss} is the small possibility that an occupied cell will not be detected. Calculate $S(R)$ using:

$$S(r) = \sum_{R} \Pr(o|A) \times (\alpha - e_{miss}). \tag{2.6}$$

More specifically, Moravec defines two update functions that approximate the sonar error, one for cells on the *range* (this is region I of (Fig. 2.1)) and another for all cells that are before region I, denoted region II [30]. Define all symbols as shown in (Table 2.1).

Putting it all together forms the equations in (Table 2.2)[30].

Table 2.1 Symbol key for Bayesian mapping formulas.

r	Distance or range returned by sensor reading
α	Angle of the sensor.
R	Maximum distance of the sensor.
β	Viewable range of sensor.
MO	Probabilistic assumption that the reading is not always correct.
n	Number of sensors.
ς_i	Range data returned by sensor $i : i = 1, 2, 3, ..., n$
H	Belief of occupation.

Table 2.2 Bayesian mapping belief functions

	Pr(Occupied) $\Rightarrow \Pr(\varsigma\|H)$	Pr(Empty) $\Rightarrow \Pr(\varsigma\|\neg H)$
Region I	$\frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2} \times MO$	$1 - \frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2} \times MO$
Region II	$1 - \frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2}$	$\frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2}$

Table 2.2 gives probabilities $\Pr(\varsigma|Occupied)$ and $\Pr(\varsigma|\neg Occupied)$ for regions I and II, but we need $\Pr(Occupied|\varsigma)$ and $\Pr(\neg Occupied|\varsigma)$. Apply Bayes formula to get [30]:

$$\Pr(H|\varsigma) = \frac{\Pr(\varsigma|Occupied)\Pr(Occupied)}{\Pr(\varsigma|Occupied)\Pr(Occupied) + \Pr(\varsigma|Empty)\Pr(Empty)} \tag{2.7}$$

The $\Pr(Occupied)$ and $\Pr(Empty)$ are the prior probabilities denoted $\Pr(H)$ and $\Pr(\neg H)$, respectively. Leading to the update rule [30]:

$$\Pr(H|\varsigma_1, ..., \varsigma_n) = \frac{\Pr(\varsigma_1, ..., \varsigma_n|H)\Pr(H)}{\Pr(\varsigma_1, ..., \varsigma_n|H)\Pr(H) + \Pr(\varsigma_1, ..., \varsigma_n|\neg H)\Pr(\neg H)} \tag{2.8}$$

$$\Rightarrow \Pr(H|\varsigma_i) = \frac{\Pr(\varsigma_i|H)\Pr(H)}{\Pr(\varsigma_i|H)\Pr(H) + \Pr(\varsigma_i|\neg H)\Pr(\neg H)} \tag{2.9}$$

Figure (Fig. 2.2) shows a three dimensional view of the sonar cone as a probability density function.

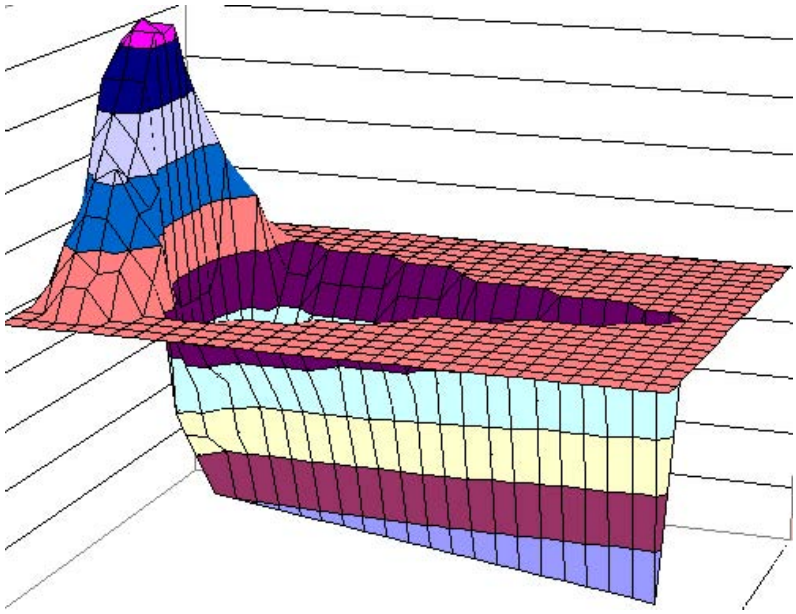


Figure 2.2 A three dimensional view the sonar cone.

2.2.2 Dempster-Shafer. Probably the most accurate and informative mapping technique without localization is the Dempster-Shafer (DS) method[30]. The DS belief theorem allows possibilities to represent conflicting evidence. That is,

$$Proposition\,\{Occupied \wedge Empty\}$$
$$\Rightarrow Frame\ of\ discernment\,\{Occupied, Empty, Don't\ Know\}$$

And the belief function must satisfy:

- Bel($\oslash$) = 0, (Define $\oslash$ as the empty set.)
- Bel($\ominus$) = 1 $\Rightarrow$ Bel(H)+Bel($\neg H$)+ Bel($\ominus$) = 1.0, ($\ominus$ is the set of all possible outcomes.)

For every non-negative integer n and every $\{A_i|1 = 1, 2, 3, ..., n\} \subseteq \ominus$:

$$Bel(A_i) \geq \sum_{I \subset \{A_1,...,A_n\}; I \neq \oslash} -1^{I+1} Bel\left(\bigcap_{i=I} A_i\right) \tag{2.10}$$

Simply stated, belief functions contributing evidence to $\ominus$ can be combined to provide a greater belief. Symbols are still defined by (Table 2.1) and the belief function is defined as shown in (Table 2.3)[30].

Table 2.3 Dempster-Shafer mapping belief functions

	m(Occupied)	m(Empty)	m(Don't Know)
Region I	$\frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2} \times MO$	0.0	$1 - \frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2} \times MO$
Region II	0.0	$\frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2}$	$1 - \frac{\frac{R-r}{R}+\frac{\beta-\alpha}{\beta}}{2}$

The update function is then [30]:

$$m(occupied) = \frac{\sum_{A_i \wedge B_j = Occupied} m(A_i)m(B_j)}{1 - \sum_{A_i \wedge B_j = \oslash} m(A_i)m(B_j)} \tag{2.11}$$

$$m(Empty) = \frac{\sum_{A_i \wedge B_j = Empty} m(A_i)m(B_j)}{1 - \sum_{A_i \wedge B_j = \oslash} m(A_i)m(B_j)} \tag{2.12}$$

$$m(Don'tKnow) = \frac{\sum_{A_i \wedge B_j = Don'tKnow} m(A_i)m(B_j)}{1 - \sum_{A_i \wedge B_j = \oslash} m(A_i)m(B_j)} \tag{2.13}$$

The conflict between readings is removed through normalization[30]:

$$Con(Bel_1, Bel_2) = \log\left(\frac{1}{\sum_{A_i \wedge B_j = \oslash} m_1(A_i)m_2(B_j)}\right) \tag{2.14}$$

Borenstein and Koren improvised the occupancy grid map idea, allowing for faster processing and making it possible for use in real time obstacle avoidance and robot navigation by approximating the probability values [4].

2.2.3 Histogram In Motion Mapping. Borenstein and Koren's approach, Histogram In Motion Mapping (HIMM)[4], represents obstacle data in a two dimensional array denoted as C. When a sonar indicates a reading for a specific cell C_{xy} in the array, C_{xy} is modified so that $C'_{xy} \leftarrow C_{xy}+3$ and all its adjacent cells $(x\pm1)(y\pm1)$ are incremented by .5. All cell values in C are limited so that $0 \leq C'_{xy} \leq 15$. Additionally, cells along the path from the robot center point (x_0, y_0) to (x, y) denoted as C_{ij} are decreased by 1 (Fig. 2.3). Notice that this technique assumes a higher

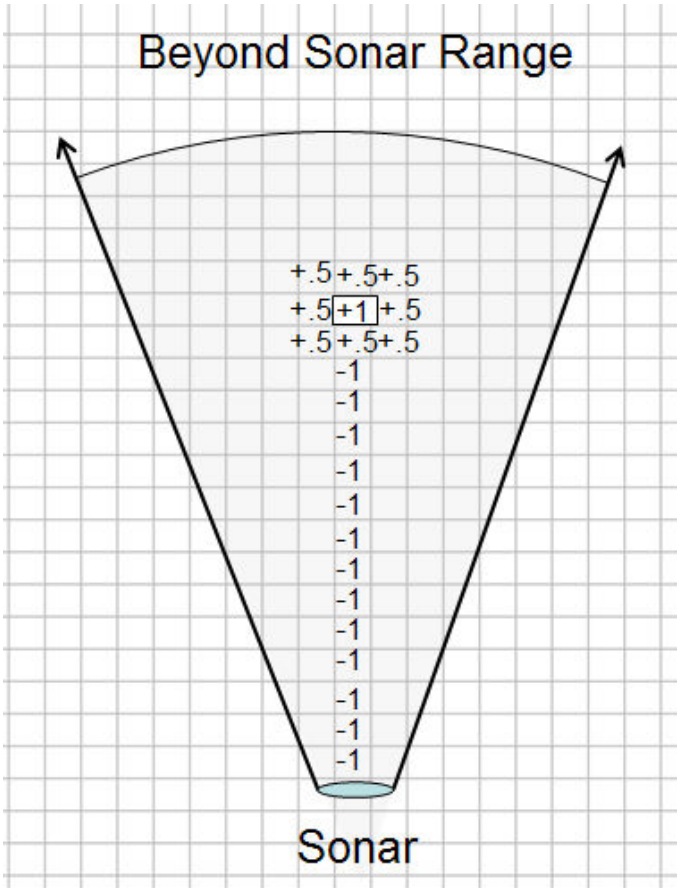


Figure 2.3 Histogram in Motion Mapping Sonar Cone

belief that a cell is occupied than not occupied. This simple method of building a map demands little processor time and memory space, so it is quickly processed by a computer.

Borenstein and Koren also introduce the idea of using a polar histogram (Fig. 2.4) for obstacle avoidance[5].

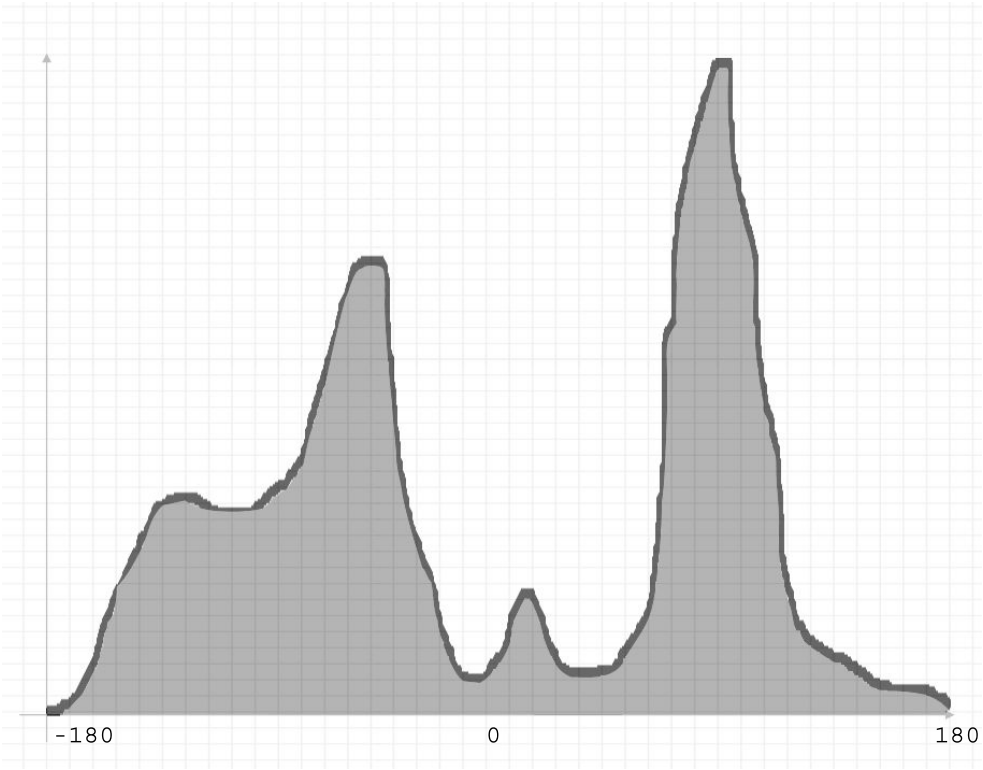


Figure 2.4 Polar Vector Field Histogram

Within a predetermined window around the robot, a polar sweep is made in 5° increments. The angle β of the current point (i, j) with respect to the vehicle center point (x_0, y_0), is determined by,

$$\beta_{i,j} = \arctan\left(\frac{j - y_0}{i - x_0}\right) \tag{2.15}$$

For each cell (i, j), magnitude is calculated by,

$$m_{ij} = C_{i,j}^2 * \kappa \tag{2.16}$$

and κ is set so that the value within C_{ij} is scaled where the closer it is to the robot the higher the value it is assigned, and the farther away, the less weight it is given.

For each cone ϕ, cells are summed to determine a magnitude vector $\vec{k}_\phi$.

$$\vec{k}_\phi \leftarrow \sum_{ij} m_{ij} \tag{2.17}$$

Determination of which cone point (i, j) belongs to is accomplished by finding β_{ij}, where $\phi - 1 < \beta_{ij} \leq \phi$ indicates point (i, j) in the cone from $(\phi - 1, \phi]$. The robot travels in the direction with the largest grouping containing lower values of $\vec{k}$.

An advantage of HIMM is its ability to progressively adapt the strength of an obstacle avoidance reaction to the level of evidence for the existence of an obstacle, and do so quickly. By using this technique which is computationally cheap, we free up processor time for use in the cognitive conversion process while providing obstacle avoidance. Another popular approach to mapping, known as Expectation Maximization (EM) [36], holds probabilistic information about the world and constructs the most likely map from it.

2.3 Expectation Maximization for Robotic Mapping and Localization

The Expectation Maximization algorithm or the concept of it has been around since the late 1950's. However, it was not put together as a complete algorithm until 1970 by Baum et al. [1]. Later in 1977, Dempster et al. [8] formalized the algorithm and coined the term Expectation Maximization.

EM is used in many scientific fields including medicine, electrical engineering, business, and computer science. Within computer science EM is used for imaging, networking, robotic mapping and localization and many other applications. Thrun, et al., [36] performed the seminal work that extends the EM algorithm to the use of mapping and localization for robotic applications using landmarks. To use the EM algorithm for mapping and localization, the map and robot path is formed as a maximum likelihood function. This function is then maximized to determine the most likely map and path traveled.

The Expectation Maximization algorithm alternates between an expectation step (E-step) and a maximization step (M-step) [8]. In the E-step, the current map is not changed but the probability distributions are calculated for past and current robot locations. In the M-step, the most likely map is computed based on the estimation result gained from the E-step [36]. Alternating both steps, the robot simultaneously improves its localization and its map, performing hill-climbing or gradient assent to lead to a local maximum in likelihood space. The probabilistic nature of the estimation algorithm makes it considerably robust to ambiguities and noise, both in the odometry and in perception. Additionally, it enables the robot to revise past location estimates with new sensor data [36].

The Expectation Maximization method used in [36] is effective but has some weaknesses. Even constraining the algorithm where only discrete probabilities are considered with a spatial resolution of 1 meter and angular resolution of 5 degrees, the processing time required for their experiments is still close to, 2 hours. Some methods to speed up processing are [36]:

- caching,
- symmetry exploitation,
- coarse-grained temporal resolution,
- selective computation, and
- selective memorization.

Explanations of each can be found in [36]. These techniques applied in his experimentations reduce computation time by approximately 2.98×10^8 [36].

2.4 Cognitive Mapping

Using a simple occupancy grid presents several problems. Of these problems, the amount of memory needed to represent a large area is of primary concern. For example, if the grid resolution is 10 centimeter by 10 centimeter cells, mapping a

100×100 meter area requires $(100 \times 100) \times (100 \times 100) \times 16_{bytes} = 1.53_{GB}$ of memory space. Performing localization on a dataset this large is computationally difficult making it important for us to represent the same information in a more memory friendly fashion.

Cognitive mapping with the use of ASRs is a topological mapping approach. That is, the world is represented more from the perspective of the robot and in some cases maintains very little metric information about the world. Instead, the map is created with consideration to where the ASRs are with respect to each other and the robot.

2.4.1 Absolute Space Representation. An Absolute Space Representation (ASR) [13], is a cognitive mapping technique used to build models of rooms or spaces visited. Absolute space comes from the idea that the representation for each space should be independent of all other spaces. Also, humans view the world cognitively and so should our robots for ease of relation.

Rooms/nodes are separated by access points/edges, that in the graph are represented as edges connecting nodes. This technique is explained by Hill, Han, and Lent [13]. While the idea of the ASR [13] has a foundation with three dimensional image based localization and mapping (much like Video SLAM [32], using video to extrapolate range data and/or landmark data), it is also applicable with respect to indoor mapping using ultrasonic sonars as the range device [20]. Figure 2.5 shows a standard Cartesian map translated to an ASR. Notice the rooms become nodes and the exits become edges connecting the nodes.

The use of ASRs becomes more necessary as the size of the mapped environment increases. By placing non-current ASR data in persistent storage, the robot must only hold a limited amount of information in main memory at any given time. Additionally, when the robot uses the cognitive map at a later time, it need only

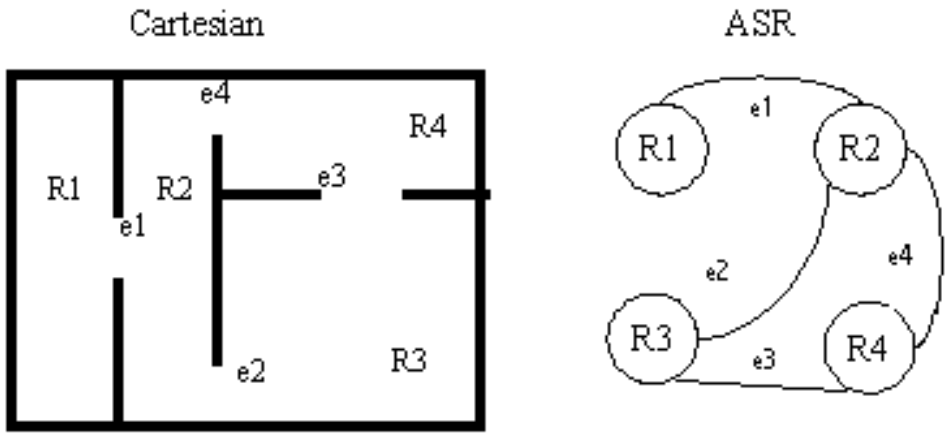


Figure 2.5 Two Room ASR Example

know which ASR it currently occupies and have the data for that ASR and its neighboring ASRs loaded into memory.

Jeffries and Yeap [17] expand the theory of cognitive maps which at its core has the notion of an autonomous agent building a map in memory, i.e. a *map in the head* termed a *cognitive map*, for the places it visits. The agent first develops a representation for each space visited [39][38]. The space occupied by the robot, termed the *local space*, is defined as the region which the robot perceives enclosing it (a room; hall, etc). The robot's cognitive map grows as the representation of each local space it visits is added to a topological network or graph of ASRs.

Jeffries and Yeap [17] initially test this approach with a simulation of an agent traveling in a complex 2D representation of a hospital environment. Data produced by this simulation includes a comprehensive description of the underlying theory and algorithms employed in its implementation. This theory is applied to the problem of an autonomous mobile robot equipped with sonar sensors, building a map from its experience of the places it has visited [17]. In this case the cognitive map consists of a *local space representation* for each local space visited with connections to other rooms that have been experienced as neighbors. All local space representations have

their own local coordinate systems and are independent of all others. Our approach builds on this method by applying the use of polylines to the representation of the ASR. This enhances our ability to use the ASR for navigation and localization.

2.4.2 Topological Mapping. It is important to note that cognitive mapping is a subset of a more general type of mapping known as topological mapping. Topological mapping, in the truest sense, is mapping where everything in the map is represented with a position reference based on something else. In a typical indoor environment a hall can be an edge connecting two rooms, and the rooms can be vertices on a graph. Jensfelt [19] provides a good survey of some topological mapping schemes. The advantage of this approach is that they can be constructed without knowing the exact geographical relation between nodes as they are an abstract construct[19].

2.4.3 Polyline Simplification. Most implementations of robot mapping represent map data as some form of occupancy grid or topological network. Our approach merges these two ideas. We maintain world data by transforming a temporary occupancy grid to a set of lines and structuring those lines in a topological network. The size of the data structure is significantly smaller using polylines in place of an occupancy grid, and using a cognitive map for path planning and localization becomes much faster as the search space is reduced. Using a polygonal representation with sonars has been previously considered infeasible, as the amount of noise generated by the sonars complicates the process greatly [11]. Latombe and Banos [11] use polylines with the help of a laser range finder, instead of sonars. The vehicle moves from point to point and performs a polar sweep at each point. Each new polar sweep is added to the current model. Lines are extracted from the raw range data using a polar line fitting algorithm. Using sonars necessitates the need for developing a new method of fitting range data to lines. We extend their system

by developing a new data fitting method and storing the data in a cognitive map for use in later localization and navigation.

2.4.4 Douglas-Peucker. It is necessary after extracting vertices from range data, to simplify and reduce vertices that are not needed to represent the map. A popular method in the mathematics community known as the Douglas-Peucker line simplification method [9] works well to reduce unneeded vertices in polylines. The Douglas-Peucker line simplification algorithm [9] uses the closeness of a vertex to an edge segment as judgment criterion for elimination. The algorithm starts with a crude guess of a simplified polyline, the single edge joining the first and last vertices of the polyline. Remaining vertices are tested for closeness to that edge. If these vertices are further than a specified tolerance from the edge, then the furthest vertex from it is added to the simplification. This process creates a new guess for the simplified polyline. Applying this process recursively for each edge until all vertices of the original polyline are within tolerance of the simplification yields the final simplified set of edges (Fig. 2.6).

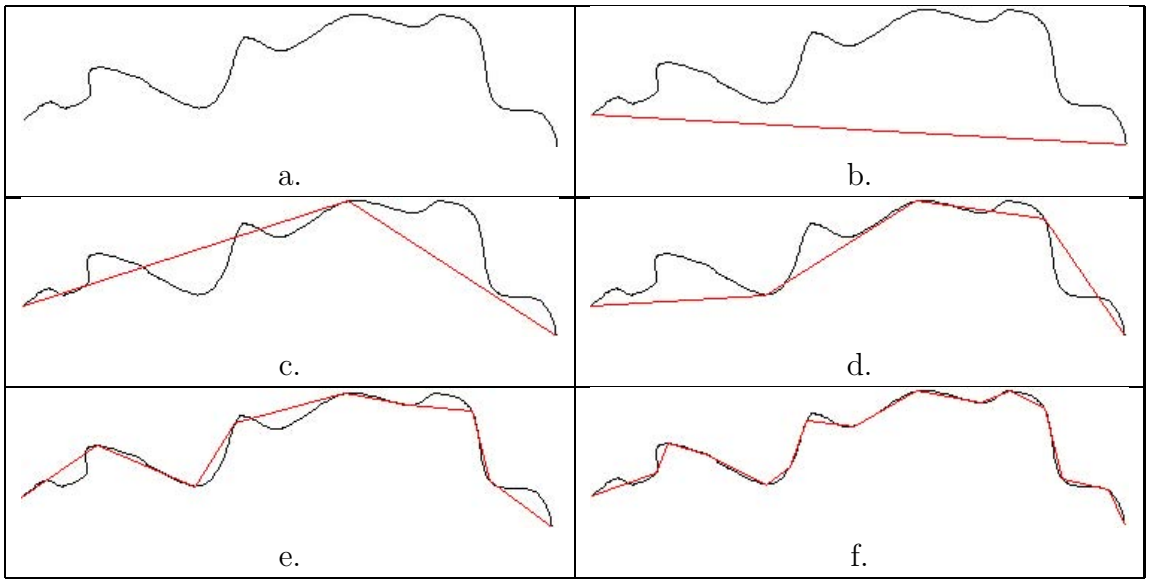


Figure 2.6 Steps in the Duaglus Peuker Algorithm

2.4.5 Edge Detection. Extracting points from the grid is a close problem to the more common edge detection problem in image processing. Consequently, there are many algorithms available which work well to accomplish this task when the edges (in our case obstacle boundaries) are well defined. Sonars produce noisy edges and so most of these algorithms are not efficient to accomplish this task. One algorithm called the *Hough Transform* [14] also works well when the edges are not well defined and is used often in robotic mapping for edge extraction [18][32]. A problem with using the *Hough Transform* is the amount of computation required finding edges. This method is general in nature and not specific to robot maps. With a map, more information is known about where edges should or should not be based on the path of the robot and areas beyond the range of the robot's sensors. This information speeds up the location and determination of edges in the map.

2.5 Localization

Monte Carlo Localization (MCL) provides a solution to the global localization and kidnapped robot problem in a highly robust and efficient way. MCL accommodates arbitrary noise distributions and therefore avoids a need to extract features from data originating from whatever sensors that may be used. These sensors can be sonars, lasers, cameras, or any other type of range detection device. We denote a belief to be the agent's certainty when it detects an object, as how sure it is of where that object is, and if it is even an object. Representing the belief is a set of samples o, drawn according to the posterior distribution over robot poses a and is the key idea of MCL. The set of particles is represented as random variable x. The MCL performs a recursive update function to find the recursive function

$$Bel(x_t) = p(o|x_t) \int p(x_t|x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{2.18}$$

where t is time.The MCL recursive update function is performed as a three stage process[37]:

1. Sample x_{t-1} from the sample set. Distribute x_t according to the belief distribution $Bel(x_{t-1})$.

2. The pair $\langle x_t, x_{t-1} \rangle$ is distributed according to the product distribution

$$q_t = p(x_t|x_{t-1}, a_{t-1}) \times Bel(x_{t-1}) \tag{2.19}$$

3. Correct the mismatch between distribution q_t and the desired distribution (Equation 2.18). The deference is accounted for using the following importance factor:

$$w = \frac{p(o_t|x_t)p(x_t|x_{t-1}, a_{t-1}) \times Bel(x_{t-1})}{p(x_t|x_{t-1}, a_{t-1}) \times Bel(x_{t-1})} = p(o_t|x_t) \tag{2.20}$$

So, rather than approximating posterior distributions in parametric form, as is the case for Kalman filter and Markov localization algorithms, MCL represents the posteriors by a random collection of weighted samples that approximate the desired distribution X. Most work estimating the state recursively, using samples, is very recent. Computer vision researchers have proposed the same algorithm as the *condensation* algorithm [37].

The MCL method based on the *condensation* algorithm is a vision-based Bayesian filtering method that uses a sampling-based density representation and can represent multi-modal probability distributions [7]. Using a visual map of the ceiling obtained by mosaicing, it localizes the robot globally using a scalar brightness measurement. Jensfelt, et al. [19] introduced modifications allowing greater efficiency in large symmetric environments. Sensor information (sonar, laser or brightness measurements) provides low feature specificity requiring that they are probabilistic and that the robot move around while probabilities gradually converge toward a localized peak [32].

Several algorithms have been developed in detail to perform localization. Neira [28] describes a recursive branch and bound algorithm called Geometric Constraints Branch and Bound (GCBB), which computes the best hypothesis H from the available measurements and the features of the stochastic map, using unary (one) and binary (two) location independent geometric constraints. The algorithm starts with an empty hypothesis and proceeds in a depth-first branch and bound manner. At the leaf-level *solution space* of the interpretation tree, the algorithm checks whether this leaf is a hypothesis having more consistent pairings than the current best. Satisfying binary geometric constraints does not guarantee that a hypothesis is globally consistent [12], the vehicle location is then estimated and a joint compatibility test [28] is used to verify global consistency of the matching hypothesis. The algorithm performs a bounded search, evaluating the potential benefit of considering the star-branch before it recurses. Branching can be done by pre-ordering the sensor measurements so that the most precise are considered first. We can pre-compute unary and binary constraints before recursion starts because they are both location independent [29].

An alternative algorithm, originally developed for geometric object recognition by Bolles, et al. [2] and Faugeras, et al.[10], follows a hypothesis generation-verification scheme. Location independent unary and binary geometric constraints are useful to generate candidate hypotheses, just as in GCBB. However, as soon as the number of pairings is adequate to determine the vehicle location (when we have two points and two non-parallel segments), the verification of the hypotheses takes place, predicting the location of map features and also searching for pairings with the remaining measurements.

III. Design

Development of a software system requires the use of a structured top down design approach starting with a problem statement and ending with an implemented system. Robotic mapping systems afford the opportunity for modular, sequential development. First, the robot must travel before it can map, therefore the guidance and control is accomplished first. Next, a mapping algorithm is picked to map as the robot moves. After the robot wandering and mapping algorithms are completed, the efficiency of the stored information is examined and optimized. The cognitive conversion algorithm is developed before moving on to localization. Localization, typically the most difficult step in performing Simultaneous Localization And Mapping (SLAM), is processor intensive, therefore it is important for the robot to localize using the simplified cognitive map.

The first floor of the Air Force Institute of Technology building (Fig. 3.1) provides an ideal testing environment for the robot mapping system and is where we perform most real world testing. Another office building is used for the small cluttered environment test however no blueprint is available for that office.

3.1 Implementation

The system developed in this thesis is comprised of several distinct algorithms. At the highest level, the system consists of the robot traveling around its environment creating a Modified Histogram in Motion Mapping (MHIMM) map. Each time the robot detects that it leaves a room (enclosed area), it creates a complete polygonal map of the previous room and saves that map to the hard drive as an Absolute Space Representation (ASR) represented by polylines. A new map is started and the process begins again in the new room. The mapping is accomplished real-time.

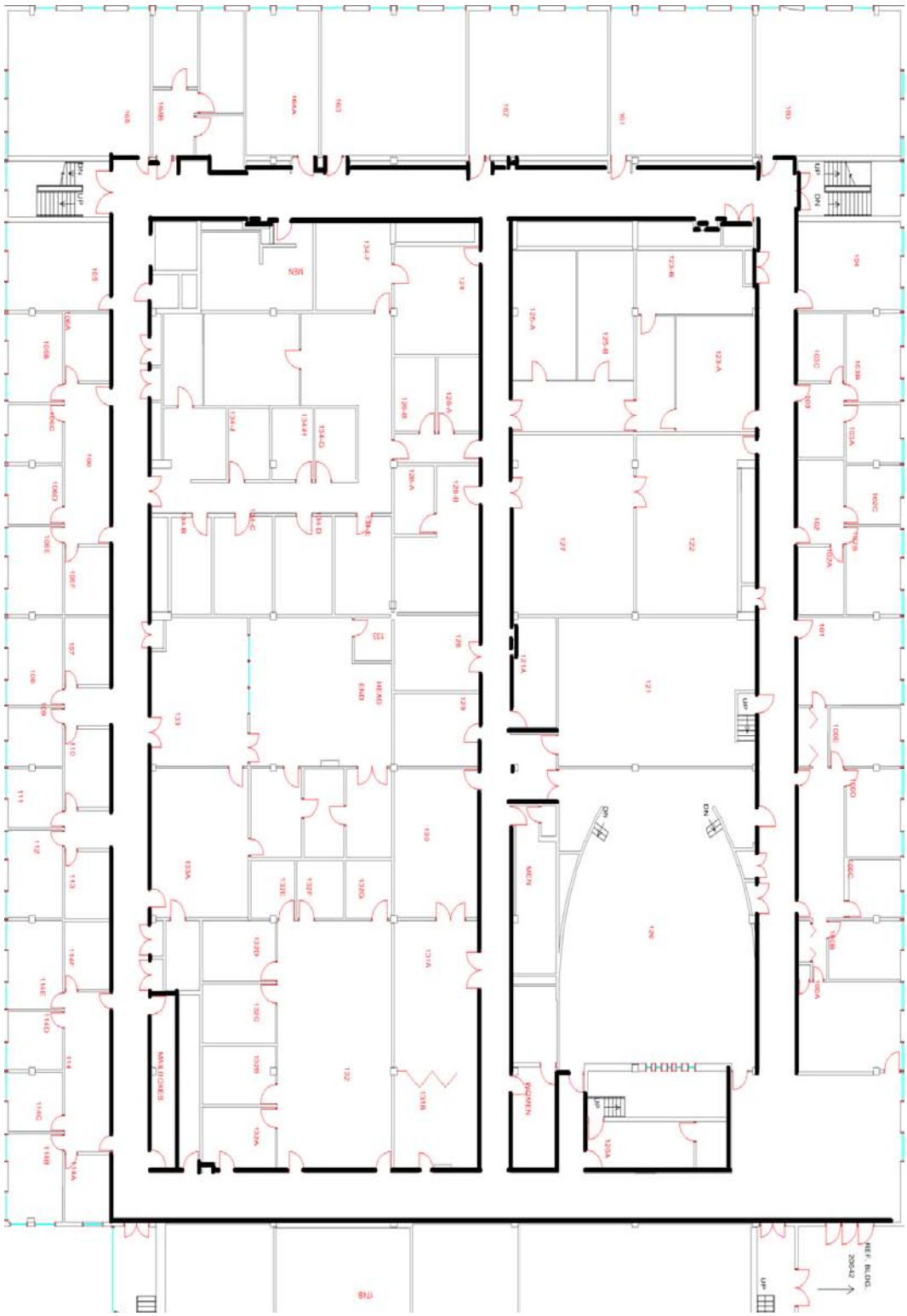
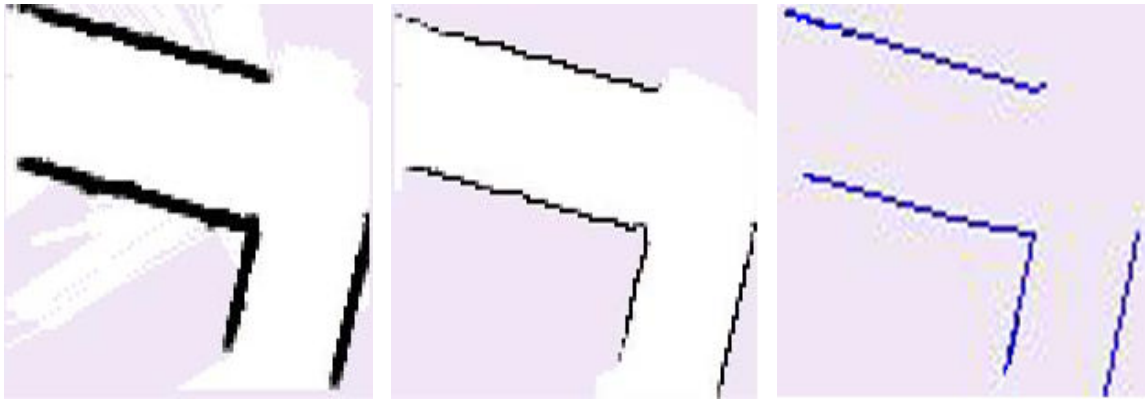


Figure 3.1 Blue print of the mapping test environment.

Creating a map of a room is a multistage process. It begins by cleaning the HIMM map (Fig. 3.2a) ($50 \times 50 = 2500$ integer matrix) leaving walls represented by no more than one cell thick lines in the grid (Fig. 3.2b) (50×50 integer matrix). Next, an extraction algorithm is called to create an ordered list of vertices ($250 \times 2 = 500$ integers) using a nearest neighbor, follow-and-remove process. Using the new list of vertices, a final representation (Fig. 3.2c) ($9 \times 2 = 18$ integers) is generated using vertex and edge reduction algorithms. Exit and entry locations are stored for later use as pivot points for rotational adjustments to compensate for localization errors. Finally, the robot combines all the ASRs to create a human readable master map.



(a) (b) (c)

Figure 3.2 Complete polyline process. The raw map is shown in (a), followed by the cleaned map (b). Finally (c) is the simplified fitted polyline.

To create this map the robot performs a two staged process. first, it fragments the map so that each discrete area or room visited can be mapped independently. Second, it converts the occupancy grid to a series of polylines that are then stored in a spatial representation.

3.1.1 Absolute Space Representation (ASR) Construction. When the robot completes the mapping of one room/space it constructs an ASR from the MHIMM map. This construction involves creating the raw MHIMM map, cleaning it, and

converting it to a cognitive map. Finally, the robot adds the map to the ASR container. We break this process into the sequence;

1. cleaning the raw HIMM map (Fig. 3.3a) to (Fig. 3.3b),
2. creating an ordered list of vertices by extracting points (Fig. 3.3c),
3. eliminating points that are close together (Fig. 3.3d),
4. reducing the number of edges (Fig. 3.3d) to (Fig. 3.3e),
5. fitting the remaining edges to a line with linear regression model (Fig. 3.3e) to (Fig. 3.3f) to (Fig. 3.3g),
6. calculating a reference point used later when all the ASRs are put back together (Fig. 3.3h),
7. and finally calculating the localization drift and adding the new ASR to the ASR container (Fig. 3.3h) to (Fig. 3.3i).

3.1.2 ASR Boundary Detection. Before mapping an ASR, the algorithm must be able to detect the beginning of a new ASR and the end of the previous ASR. The traversal from one room to another can be thought of as a series of states. In a room, the robot typically will be close to no walls, one wall, or two walls that are orthogonal to each other. Passing from one room to another, the robot passing an access point will detect obstacles close to it, one on each side of the robot. All these possibilities present the following:

- The robot is in a room.
- The robot is at an access point.

The following events are used to transition among states ($\neg$ is the negation symbol, $\wedge$ represents *and*, and $\vee$ represents *or*.):

- An obstacle is close to the left of the robot, denoted (L).

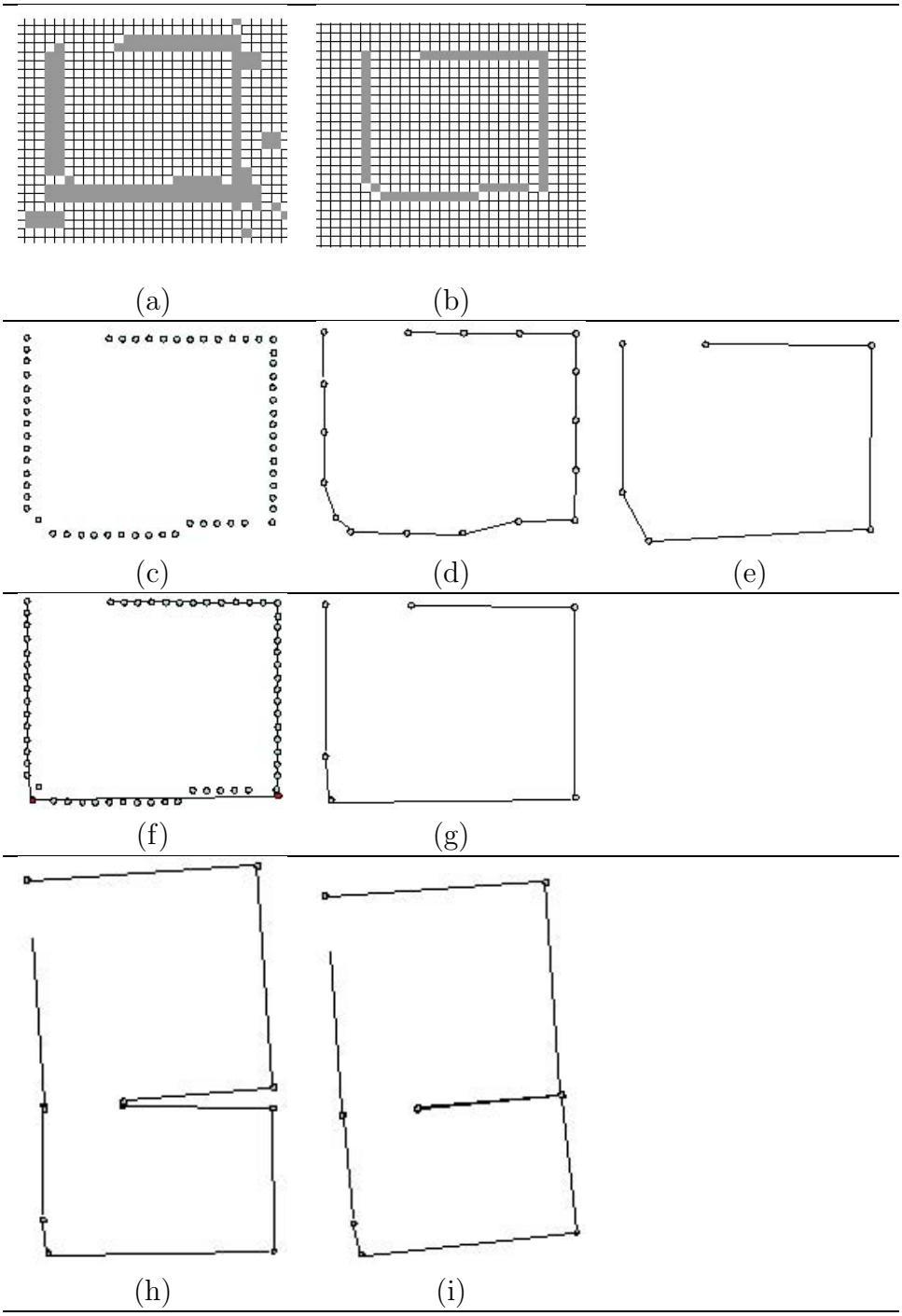


Figure 3.3 Steps in the ASR creation process. These steps involve cleaning the raw HIMM map (a) which yields (b), then points are extracted creating an ordered list of vertices (c), then simplified (d) and (e). These points are fit to a line with a linear regression model (f) and (g). Finally, the ASR is added to the map (h) and localized (i).

- An obstacle is not close to the left of the robot, denoted (¬L).
- An obstacle is close to the right of the robot, denoted (R).
- An obstacle is not close to the right of the robot, denoted (¬R).

If the robot travels from a room to an access, as it enters a new room an exit is detected at the access point. This is shown as a state machine in (Fig. 3.4). Hallways are not considered in this algorithm. In later testing when hallways are mapped ASRs are constructed based on the distance the robot travels; a new ASR is created every 4.5 meters.

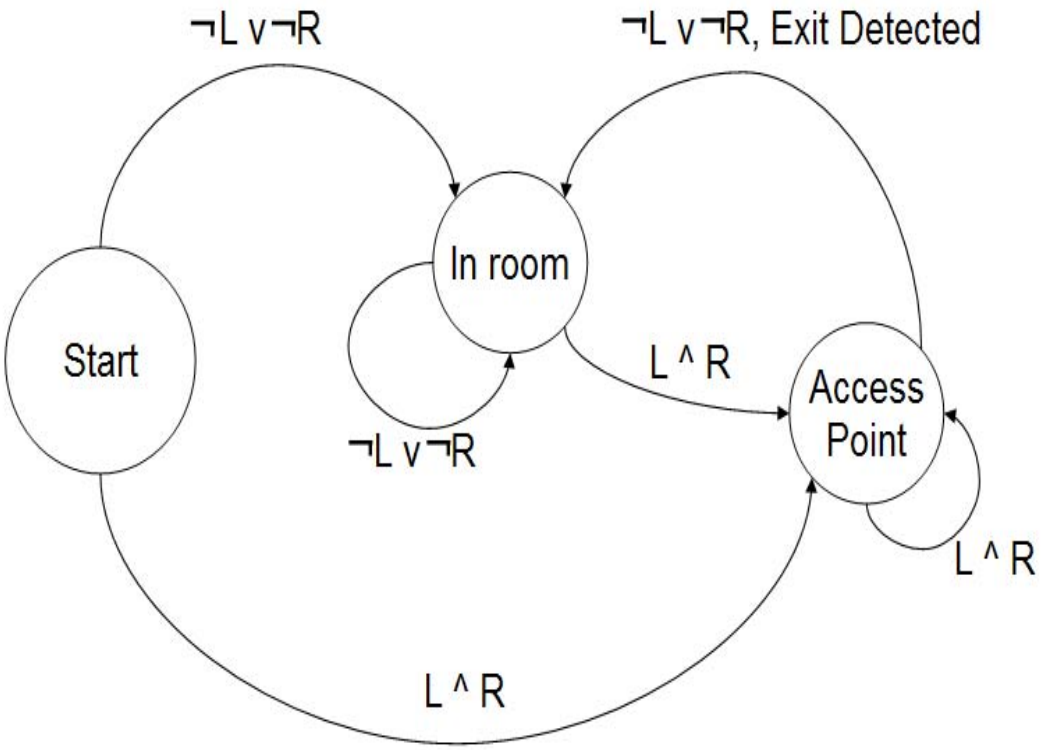


Figure 3.4 A state machine indicating the exit detection process.

3.1.3 Building a Mapping Algorithm. Section 2.2 gives the definitions shown in (Table 3.1) that are used in this section to develop a general algorithm for grid mapping.

Table 3.1 Symbol key for the general occupancy grid mapping algorithm.

Symbol	**Definition**
$\wp$	Scalar used to determine cell size.
C	2 dimensional matrix that represents the world.
m	Max distance divided by $\wp$ expected in the x direction.
n	Max distance expected to travel in the y direction divided by $\wp$.
h	Robots direction.
sh_i	Each sonar's angle relative to the robot. $i = \{1, 2, 3, ...number\ of\ sonars\}$
h_i	The global angle of each sonar. $h_i = h + sh_i$.

Sensors are not at the center of the robot and the mapping requires the location of obstacles relative to the center of the robot. Each sonar has an x offset and y offset from the robot center point, denoted sox_i and soy_i respectively. This offset must be transformed from the robots local coordinate system to the global coordinate system. This gives us a modification to (Equation 2.1):

$$x_i = \frac{r_i \times \cos(h_i) + x_{offset_i}}{\wp} \tag{3.1}$$

$$y_i = \frac{r_i \times \sin(h_i) + y_{offset_i}}{\wp} \tag{3.2}$$

As expressed in [31] the relationship of a point (x, y) and its transformed point (x', y') given a circular rotation of θ can be expressed as:

$$\begin{aligned} \begin{pmatrix} x' \\ y' \end{pmatrix} &= \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ &= \begin{pmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \end{pmatrix} \end{aligned} \tag{3.3}$$

and our offset for each sonar is:

$$x_{offset_i} = x' = sox_i \cos(h) - soy_i \sin(h) \tag{3.4}$$

$$y_{offset_i} = y' = sox_i \sin(h) + soy_i \cos(h) \tag{3.5}$$

substituting this into (Equation 3.1) and (Equation 3.2) and adding the offset for the robots current pose (x_0, y_0) gives:

$$x_i = \frac{r_i \times \cos(h_i) + (sox_i \cos(h) - soy_i \sin(h)) + x_0}{\wp} \tag{3.6}$$

$$y_i = \frac{r_i \times \sin(h_i) + (sox_i \sin(h) - soy_i \cos(h)) + y_0}{\wp} \tag{3.7}$$

Figure 3.5 shows a transformation of a robot in a global map C where a sonar detects an obstacle at point p that is transformed to point p' to adjust for the misalignment in the sonars x and y offsets caused by the difference in the robots heading and the world frame.

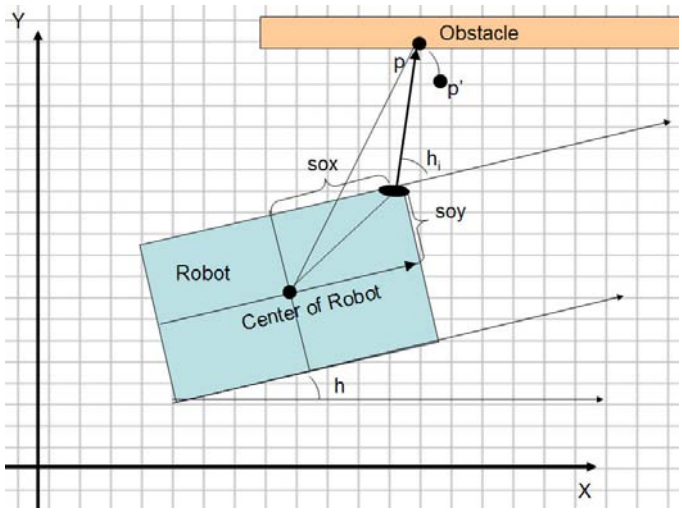


Figure 3.5 Local coordinate system of a robot in a global map.

Mapping is then a matter of cycling through all the sonars for each sonar scan and updating C, the map (Fig. A.1).

3.1.4 Modified Histogram In Motion Mapping Algorithm. The basis of the mapping algorithm is the underlying modified version of the Histogram In Motion

Mapping (HIMM)[4]. Following is the development of the HIMM algorithm and from it, modifications needed for this system.

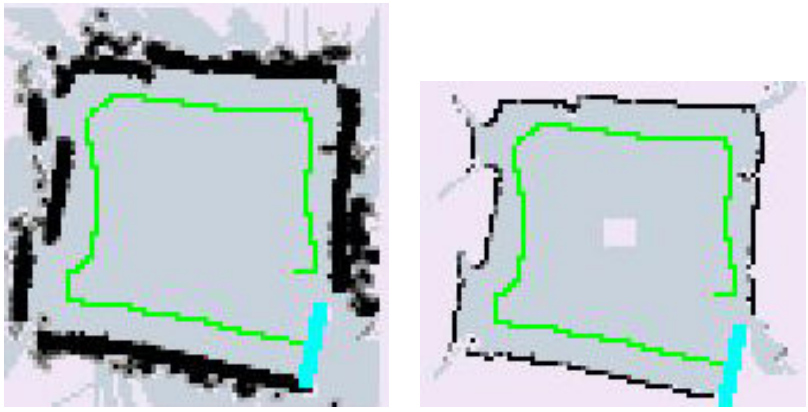
From Section 2.2.3, obstacle data in a histogram is represented in a two dimensional array denoted C. The active spot of a sonar reading (x, y), $C'_{xy} \leftarrow C_{xy} + 1$ and all its surrounding cells $(x \pm 1)(y \pm 1)$ are incremented by .5. All cell values in C are limited so that $0 \leq C'_{xy} \leq 15$. Additionally, cells along the path from the robot center point (x_0, y_0) to (x, y) denoted as C_{ij} are decreased by 1 (Fig. 2.3). Moreover, the further away an obstacle is detected by a sonar, the less accurate that reading is as it moves further out in the sonar cone (Fig. 2.3). Because sonar data degrades with distance, only points within a window $\{w : x_0 - w$ to $x_0 + w$ and $y_0 - w$ to $y_0 + w\}$ around the robot are considered. Remember that each sonar has an x offset and y offset from the robot center point, denoted sox_i and soy_i respectively. Range data returned from the Pioneer robots' sonars r_i are in millimeters and setting $\wp$ to 10_{cm} works well. Inserting the HIMM technique for the update function of the occupancy grid mapping algorithm in (Appendix A, Fig. A.1) results with the algorithm shown in (Appendix A, Fig. A.2).

The need to make more defined inside edges and information about the robot path prompts a few small modifications leading to our Modified Histogram In Motion Mapping (MHIMM) algorithm. These modifications are:

1. For all updates the vehicle center point (x_0, y_0) is set to -100 to mark the path of the robot. Later the robot will need this information to clean the map.
2. Empty regions are permitted to drop below 0 to -3. For all x and y, $-3 \leq C_{xy} \leq 15$. This distinguishes explored from unexplored areas.
3. If the sonar range is greater then w then all cells in the direction of that sonar to the window boundary are decreased by 1. This modification filters out close proximity noise considerably.

The resulting algorithm is shown in (Appendix A, Fig. A.3).

3.1.5 Clean Map Algorithm. From the MHIMM built occupancy grid map, our cleanup algorithm incorporates some modifications of the original mapping technique provided [4]. In their article, empty cells $empty(C_{xy})$ are decreased by -1 but not allowed to drop below 0. To increase the difference between occupied and unoccupied cells, the lower bound of C is decreased so that $empty(C_{xy}) \geq -3$ allowing us to distinguish between an unexplored region and an empty region. More importantly, spaces occupied by the robot itself are marked as $robot_{path}$ (-100) for use in the map cleaning algorithm (the update function is prevented from modifying this value). This is shown in (Fig. 3.6a) and the modified map demonstrates a cleanly mapped region inside the empty space and messy area in the unexplored region (Fig. 3.6b).



(a) (b)

Figure 3.6 A HIMM before (a) and after (b) applying the HIMM cleaning algorithm. The line inside the room represents the path of the robot.

The cleaning process is straight forward; any cell deemed empty or occupied is checked to see if a ray from it to the path of the robot is possible. If the ray traverses occupied cells then it is set to an unexplored or unknown state (Fig. 3.7). We represent the map to be cleaned as a matrix H where $H_{x,y}$ is a specific position of the map. The final filtered map is H'. Let max_{dist} be the maximum effective range of the sonars. Part of our algorithm requires creating a ray from every point (x, y) where $H_{x,y} > 0$ to $x \pm max_{dist}, y \pm max_{dist}$. All line functions use Bresenham's line algorithm

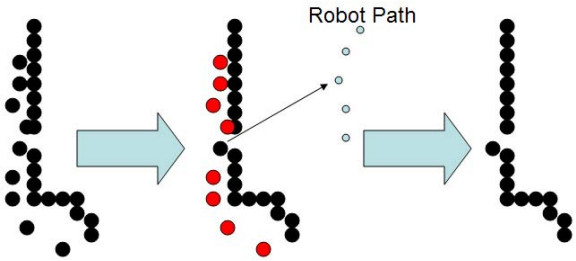


Figure 3.7 Any occupied cell not able to draw a ray to the path that the robot traveled without going through another occupied cell or it exceeds the max effective distance of a sonar, it is discounted as noise.

[6] for efficiency. To represent the line algorithm, we define $\Gamma(p_a, p_b, \{start, next\})$ which returns points along the line from p_a to p_b. Finally, ϵ is a threshold value where $H_{x,y} > \epsilon$ is considered occupied. The full algorithm is shown in (Fig. 3.8).

3.1.6 Point Extraction Algorithm. One of the big challenges is taking point data from the two dimensional array and turning it into a list of organized vertices, V. The points must be retrieved in such a fashion that they form organized polylines, the point on the map that the edge extraction starts being arbitrary. Simply adding vertices to a list and treating them as a polyline results in a jumble of seemingly random lines. A point in H is only added to the vertex list V if $H_{x,y} > \epsilon$. Consecutive vertices creates segments that start at point p_s and end at p_e.

In general, the algorithm retrieves points by following occupied cells, adding the point to V and removing it from H until $|H| = 0$. This algorithm is detailed in (Fig. 3.9).

While retrieving points on the map, it is important to identify where one polyline ends and another begins. To do this, a threshold variable γ, is defined as the maximum allowable distance between two cells containing no occupied spaces. If the distance is greater then γ, a break is inserted denoting the end of a polyline.

3.1.7 Edge Simplification Algorithm. Before we perform any edge simplification, it is important to reduce unnecessary vertices first. The process of simplifying

```
CleanMap(H)
  For (every point (x, y) of H)
    If(H_{x,y} > ε)
      p_a ← (x, y)
      found ← false
      For (−max_dist to +max_dist as j)
        For (−max_dist to +max_dist as i)
          p_b ← ((x + i), (y + j))
          p_t ← Γ(p_a, p_b, start)
          While (p_t ≠ p_b)
            p_t ← Γ(p_t, p_b, next)
            If(H_{p_t} = robot_path)
              found ← true
              H'_{p_a} ← H_{p_a}
              Break While loop
            If(H_{p_a} > ε)
              Break While loop
          If (found) Break For loop
        If (found) Break For loop
  End For
  H ← H'
```

Figure 3.8 Algorithm to clean a modified HIMM map

the list of vertices V extracted from the map begins by eliminating clusters of closely grouped points (Fig. 3.10). This is done to reduce the size of the vertex list sent to the edge simplification algorithm. While not required, the vertex simplification is simple and quick, and speeds up the overall line simplification process.

A tolerance value τ is defined such that for any point V_i, if $\Delta(V_i, V_{i+m}) < \tau$ for $m = \{i, i+1, i+2, i+3, ...n\}$, then V_{i+m} is removed from V. When $\Delta(V_i, V_{i+m}) > \tau$ then i is set $i \leftarrow i + m$. Refer to (Appendix A, Fig. A.4) for pseudo code of this algorithm. The Douglas-Peucker algorithm [9] uses the closeness of a vertex to an edge segment as judgment criterion for elimination. It starts with a crude initial guess at a simplified polyline, i.e. the single edge joining the first and last vertices of the polyline. The remaining vertices are tested for closeness to that edge. If these vertices are further than a specified tolerance from the edge, then the furthest

```
ExtractPoints(V, H)
  For (0 to |H_y| as j)
    For (0 to |H_x| as i)
      If (H_{i,j} > ε)
        p_s ← (i, j)
        p_e ← p_s
        j ← |H_y|
        i ← |H_x|
  DO
      V ← V ∪ {p_s}
      H_{p_sx, p_sy} ← 0
      p_s ← Next closest occupied cell
      If (Δ(p_s, p_e) > γ))
        Mark point this as the end of a polyline
      p_e ← p_s
  UNTIL (|H| = 0)
```

Figure 3.9 ExtractPoints

vertex from it is added to the simplification. This process creates a new guess for the simplified polyline. Applying this process recursively for each edge until all vertices of the original polyline are within tolerance of the simplification yields the final simplified set of edges.

More specifically, two endpoints of a polyline are connected with a straight line as the initial rough approximation of the polyline. Computing the distances from

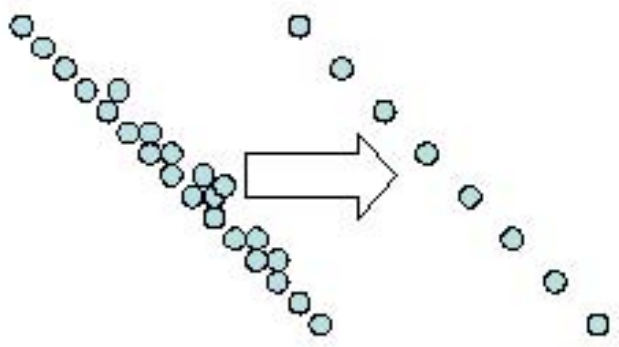


Figure 3.10 Vertex simplification process.

all intermediate polyline vertices to that (finite) line segment, allows us to judge how well it approximates the whole polyline. If all these distances are less than a specified tolerance η, then the approximation is acceptable and the endpoints are retained while the other vertices are eliminated. If any of these distances exceed η, the approximation is not considered good enough to use. In this case, the point that is furthest away is selected as a new vertex subdividing the original polyline into two shorter polylines [9]. The simplify function is initialized with $W_0 = W_n = true$, where W_0 is the first vertex and W_n is the last vertex. Let Wo be the set of output vertices. The full algorithm is shown in (Appendix A, Fig. A.5).

3.1.8 Line Fitting. The final step of the process involves using vertices returned by the Douglas-Peucker algorithm as end points to fit a *linear regression line* [25] of the form $y = b_0 + b_1 x$ using the *least squares* method where

$$b_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{3.8}$$

$$b_0 = \bar{y} - b_1 \bar{x} \tag{3.9}$$

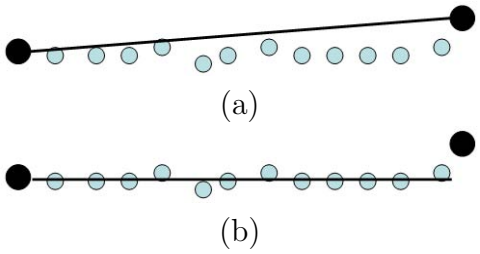
Letting $\bar{x}$ and $\bar{y}$ be the averages of the x and y points respectively.

Pearson's correlation coefficient [25] $\hat{\rho}$ is calculated to determine line suitability for a linear fit and if not, reject the fitted line. The coefficient $\hat{\rho}$ is represented as:

$$\hat{\rho} = \frac{n\sum xy - \sum x \sum y}{\sqrt{\left(n\sum x^2 - \left(\sum x\right)^2\right)\left(n\sum y^2 - \left(\sum y\right)^2\right)}} \tag{3.10}$$

A line with $|\hat{\rho}| < .9$ (testing showed .9 as a value which produced the best results) where $-1 < \hat{\rho} < 1$, is classified as a bad correlation in our implementation. The Douglas-Peucker algorithm restricts this from allowing small noise data to produce large errors in the output map (Fig. 3.11) requiring this extra fitting process. This

process will work equally well if the mapped wall is straight or curved. The Douglas-Peucker algorithm produces more vertices when simplifying a rounded environment then it does in a symmetric environment. The increased vertices allow the map to maintain its curved geometric data.



(a)

(b)

Figure 3.11 Line fitting before(a) and after(b) application of a linear regression fit. The points obtained from the Douglas-Peucker algorithm are used as end points in the fitting process.

3.1.9 Localization Algorithm. Localization is a computationally expensive task, especially if localization is performed using the full map and pose dataset. To overcome this time/complexity issue, the robot performs localization on the simplified cognitive map which in turn corrects map errors. Localization is accomplished using the Expectation Maximization (EM) algorithm finding drift error for each ASR and performed iteratively while the robot maps.

3.1.9.1 Expectation Maximization. The Expectation Maximization (EM) algorithm [8] is used often in computer science to iteratively calculate the maximum likelihood of some distribution with incomplete or missing data.

Dempster et al., [8] postulates a family of sampling densities as:

$$\theta(\vec{x}|\phi) = \int_{q(x)} f(\vec{q}|\phi)d\vec{q} \tag{3.11}$$

The EM algorithm seeks to find parameters ϕ that maximizes $\theta(\vec{x}|\phi)$ with observation $\vec{x}$. This is done in two steps, an Expectation step (E-step) and Maximization-step (M-Step), iteratively converging to the maximize likelihood of $f(\vec{q}|\phi)$.

In section 3.1.9.2, we develop a model of the Expectation Maximization algorithm as a mixture model of the sonar and dead-reckoning sensors using [8][24].

3.1.9.2 Statistical Model of the EM Algorithm for Localization. We have a set of parameters $\theta_k = (\pi_k, \phi_k)$ that defines a probability distribution function of the robot's location, where $k = s, m$; s being the abstracted sonar data detailed in Section 3.1.10 and m the motor data of the probability distribution function.

E-step: During the E-step, we develop the expected complete log-likelihood function, $Q(\theta, \theta', (x_1, x_2, x_3, ..., x_N))$ based on the previous guess θ' and the data set $(x_1, x_2, x_3, ..., x_N)$ which is the combined data $\{range \cup motor\}$. This step produces a new better guess of the robot's pose $\widehat{\mu_m}$ and sensor data $\widehat{\mu_s}$ and is also used for the calculation of $\widehat{\sigma_k}^2$ and $\widehat{\pi}_k$.

Let Q be the expected value of the log-likelihood given the data $(x_1, ..., x_N)$ and $\phi_k = (\mu_k, \sigma_k)$.

$$Q(\theta, \theta', (x_1, ..., x_N)) = E\left[\sum_{i=1}^{N}\sum_{k}\log\left(\Pr(q_i^k, x_i; \theta)\right) | x = (x_1, ..., x_N)\right]$$

$$= \sum_{i=1}^{N}\sum_{k}\sum_{q}\log\left(\Pr(q_i^k, x_i; \theta)\right)\Pr(q_i^k | x_i; \theta')$$

$$= \sum_{i=1}^{N}\sum_{k}\sum_{q} q_i^k \log(\pi_k \Pr(x_i|\phi_k))\Pr(q_i^k | x_i; \theta')$$

$$= \sum_{i=1}^{N}\sum_{k} < q_i^k > \log(\pi_k p(x_i|\phi_k)) \tag{3.12}$$

where,

$$< q_i^k >= \sum_q q_i^k \Pr(q_i^k | x_i; \theta')$$

$$= \frac{\sum_q \Pr(x_i | q_i^k; \theta') \Pr(q_i^k) q_i^k}{\Pr(x_i; \theta')}$$

$$= \frac{\pi' p(x_i | \phi'_k)}{\sum_k \Pr(x_i | q_i^k; \theta') \Pr(q_i^k)}$$

$$= \frac{\pi' p(x_i | \phi'_k)}{\sum_k \pi'_k p(x_i | \phi'_k)} \tag{3.13}$$

Substituting our calculation for the M-Step (Eq. 3.17) into (Eq. 3.13) we get

$$< q_i^k >= \frac{\pi_{k'} \frac{\exp\left(-\frac{(x_i - \mu_{k'})^2}{2\sigma_{k'}^2}\right)}{\sqrt{(2\pi)}\sigma_{k'}}}{\pi'_s \frac{\exp\left(-\frac{(x_i - \mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i - \mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} \tag{3.14}$$

Substituting (Eq. 3.14 and 3.16) into (Eq. 3.12) we get

$$Q(\theta, \theta', (x_1, ..., x_N)) = \sum_{i=1}^{N} \left(\frac{\pi'_s \frac{\exp\left(-\frac{(x_i - \mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} \log\left(\pi_s \frac{\exp\left(-\frac{(x_i - \mu_s)^2}{2\sigma_s^2}\right)}{\sqrt{(2\pi)}\sigma_s}\right)}{\pi'_s \frac{\exp\left(-\frac{(x_i - \mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i - \mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} + \right.$$
$$\left. \frac{\pi'_m \frac{\exp\left(-\frac{(x_i - \mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}} \log\left(\pi_m \frac{\exp\left(-\frac{(x_i - \mu_m)^2}{2\sigma_m^2}\right)}{\sqrt{(2\pi)}\sigma_m}\right)}{\pi'_s \frac{\exp\left(-\frac{(x_i - \mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i - \mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} \right) \tag{3.15}$$

M-step: Find the value of θ that maximizes Q. Assuming that sensor noise and motor error fit a Guassian distribution, it follows:

$$\Pr(x|\mu,\sigma^2) = \frac{\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}{\sqrt{(2\pi)}\sigma} \tag{3.16}$$

$$\begin{aligned}\sum_k p(x_i|\phi_k) &= p_s(x_i|\mu_s,\sigma_s^2) + p_m(x_i|\mu_m,\sigma_m^2) \\ &= \frac{\exp\left(-\frac{(x-\mu_s)^2}{2\sigma_s^2}\right)}{\sqrt{(2\pi)}\sigma_s} + \frac{\exp\left(-\frac{(x-\mu_m)^2}{2\sigma_m^2}\right)}{\sqrt{(2\pi)}\sigma_m}\end{aligned} \tag{3.17}$$

Add the Lagrange multiplier λ to (Eq. 3.12), we optimize,

$$Opt\left(\sum_{i=1}^{N}\sum_k < q_i^k > \log\left(\pi_k p\left(x_i|\phi_k\right)\right) + \lambda\left(1 - \sum_k \pi_k\right)\right), \tag{3.18}$$

and take the partial derivatives with respect to π_k and $\phi_k = (\mu_k, \sigma_k)$ to get,

$$\frac{\partial Q}{\partial \pi_k} = \sum_{i=1}^{N}\frac{< q_i^k >}{\pi_k} - \lambda \quad \Rightarrow \quad \widehat{\pi_k} = \frac{\sum_{i=1}^{N} < q_i^k >}{N} \tag{3.19}$$

$$\frac{\partial Q}{\partial \mu_k} = \sum_{i=1}^{N}\frac{< q_i^k > (x_i - \mu_k)}{\sigma_k^2} = 0$$

$$\Rightarrow \quad \sum_{i=1}^{N}\frac{< q_i^k > x_i}{\sigma_k^2} - \sum_{i=1}^{N}\frac{< q_i^k > \mu_k}{\sigma_k^2} = 0$$

$$\Rightarrow \quad \sum_{i=1}^{N}\frac{< q_i^k > \mu_k}{\sigma_k^2} = \sum_{i=1}^{N}\frac{< q_i^k > x_i}{\sigma_k^2}$$

$$\Rightarrow \quad \frac{\mu_k}{\sigma_k^2}\sum_{i=1}^{N} < q_i^k >= \frac{1}{\sigma_k^2}\sum_{i=1}^{N} < q_i^k > x_i$$

$$\Rightarrow \quad \mu_k \sum_{i=1}^{N} < q_i^k >= \sum_{i=1}^{N} < q_i^k > x_i$$

$$\Rightarrow \quad \widehat{\mu_k} = \frac{\sum_{i=1}^{N} < q_i^k > x_i}{\sum_{i=1}^{N} < q_i^k >} \tag{3.20}$$

$$\frac{\partial Q}{\partial \sigma_k} \;=\; \sum_{i=1}^{N} \frac{< q_i^k > [(x_i - \mu_k)^2 - \sigma_k^2]}{\sigma_k^3} = 0$$

$$\Rightarrow \quad \sum_{i=1}^{N} \frac{< q_i^k > (x_i - \mu_k)^2}{\sigma_k^3} - \sum_{i=1}^{N} \frac{< q_i^k > \sigma_k^2}{\sigma_k^3} = 0$$

$$\Rightarrow \quad \sum_{i=1}^{N} \frac{< q_i^k > (x_i - \mu_k)^2}{\sigma_k^3} = \sum_{i=1}^{N} \frac{< q_i^k >}{\sigma_k}$$

$$\Rightarrow \quad \frac{1}{\sigma_k}\sum_{i=1}^{N} < q_i^k >= \frac{1}{\sigma_k^3}\sum_{i=1}^{N} < q_i^k > (x_i - \mu_k)^2$$

$$\Rightarrow \quad \frac{\sigma_k^3}{\sigma_k} = \frac{\sum_{i=1}^{N} < q_i^k > (x_i - \mu_k)^2}{\sum_{i=1}^{N} < q_i^k >}$$

$$\Rightarrow \quad \widehat{\sigma_k}^{\,2} = \frac{\sum_{i=1}^{N} < q_i^k > (x_i - \widehat{\mu_k})^2}{\sum_{i=1}^{N} < q_i^k >}. \tag{3.21}$$

Substituting (Eq. 3.14) into (Equations 3.19, 3.20 and 3.21) we get,

$$\widehat{\pi_k} = \frac{\sum_{i=1}^{N} \left(\frac{\pi_{k'} \frac{\exp\left(-\frac{(x_i-\mu_{k'})^2}{2\sigma_{k'}^2}\right)}{\sqrt{(2\pi)}\sigma_{k'}}}{\pi'_s \frac{\exp\left(-\frac{(x_i-\mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i-\mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} \right)}{N} \tag{3.22}$$

$$\widehat{\mu_k} = \frac{\sum_{i=1}^{N} \left(\frac{\pi_{k'} \frac{\exp\left(-\frac{(x_i-\mu_{k'})^2}{2\sigma_{k'}^2}\right)}{\sqrt{(2\pi)}\sigma_{k'}}}{\pi'_s \frac{\exp\left(-\frac{(x_i-\mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i-\mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} x_i \right)}{\sum_{i=1}^{N} \left(\frac{\pi_{k'} \frac{\exp\left(-\frac{(x_i-\mu_{k'})^2}{2\sigma_{k'}^2}\right)}{\sqrt{(2\pi)}\sigma_{k'}}}{\pi'_s \frac{\exp\left(-\frac{(x_i-\mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i-\mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} \right)} \tag{3.23}$$

$$\widehat{\sigma_k}^2 = \frac{\sum_{i=1}^{N} \left(\frac{\pi_{k'} \frac{\exp\left(-\frac{(x_i-\mu_{k'})^2}{2\sigma_{k'}^2}\right)}{\sqrt{(2\pi)}\sigma_{k'}}}{\pi'_s \frac{\exp\left(-\frac{(x_i-\mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i-\mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} (x_i - \widehat{\mu_k})^2 \right)}{\sum_{i=1}^{N} \left(\frac{\pi_{k'} \frac{\exp\left(-\frac{(x_i-\mu_{k'})^2}{2\sigma_{k'}^2}\right)}{\sqrt{(2\pi)}\sigma_{k'}}}{\pi'_s \frac{\exp\left(-\frac{(x_i-\mu_{s'})^2}{2\sigma_{s'}^2}\right)}{\sqrt{(2\pi)}\sigma_{s'}} + \pi'_m \frac{\exp\left(-\frac{(x_i-\mu_{m'})^2}{2\sigma_{m'}^2}\right)}{\sqrt{(2\pi)}\sigma_{m'}}} \right)}. \tag{3.24}$$

The E and M steps of the EM algorithm are repeated until convergence.

1. E-Step: Compute $Q(\theta, \theta', (x_1, ..., x_N)) = \sum_{i=1}^{N} \sum_k < q_i^k > \log(\pi_k p(x_i|\phi_k))$
2. M-Step: Find $\left\{\widehat{\pi_k}, \widehat{\mu_k}, \widehat{\sigma_k}^2\right\}$
3. Repeat until convergence.

The pseudo code for the EM algorithm can be found in Appendix A, Section A.3.

3.1.10 Line Direction. To calculate s from each ASR, a direction for each line within an ASR is calculated to obtain an ASR direction (Fig. 3.12). Each ASR

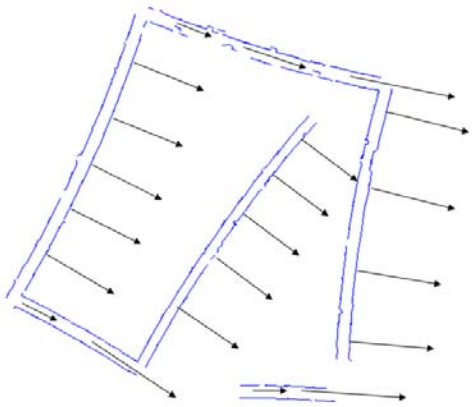


Figure 3.12 ASR directions needed to perform localization correction.

contains a list of vertices representing the polylines that make up the ASR. Every two vertices is a line segment and the direction of each line segment is obtained by treating it as a vector and calculating the angle of the vector. Slight variations in these angles (Fig. 3.13), caused by noise or obstacles, results in error that is accounted for using the EM algorithm, as described in the previous section. Let

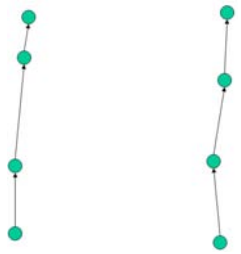


Figure 3.13 Vectors shown for a single ASR.

A denote the current vector and B the vector that comes before A. The origin of A is the termination of B and normalization of A is accomplished by setting $A_x \leftarrow A_x - B_x$ and $A_y \leftarrow A_y - B_y$. To find the angle of A let U be the normal vector along the positive x axis of the cartesian map represented by the current ASR, use:

$$\phi = \cos^{-1}\left(\frac{A \bullet U}{|A| \times |U|}\right) \tag{3.25}$$

Because U is the unit vector along the x axis, (Equation 3.25) simplifies to:

$$\phi = \cos^{-1}\left(\frac{A_x \times 1 + A_y \times 0}{|A| \times 1}\right) = \cos^{-1}\left(\frac{A_x}{|A|}\right). \tag{3.26}$$

To get a general direction for an ASR we need to normalize all vector angles within the ASRs so that we only consider a vector to be pointing in quadrants I (0 to 90 degree) and IV (0 to -90 degrees). Additionally, each angle is only considered as it's offset from the positive x axis. Because of the generally symmetric and perpendicular nature of indoor environments most vectors are parallel or normal to each other; so rather than creating a set containing two distributions (one at around ninety degrees and another around zero degree) any angle that is greater then forty five degrees is decreased by ninety degrees to acquire its normal angle. These steps are detailed in the following text.

Initially $0 \leq \phi \leq 180$ but this does not tell us if the angle is negative or positive. To determine negative angles, if $A_x < 0$ use:

$$\phi \leftarrow -\phi. \tag{3.27}$$

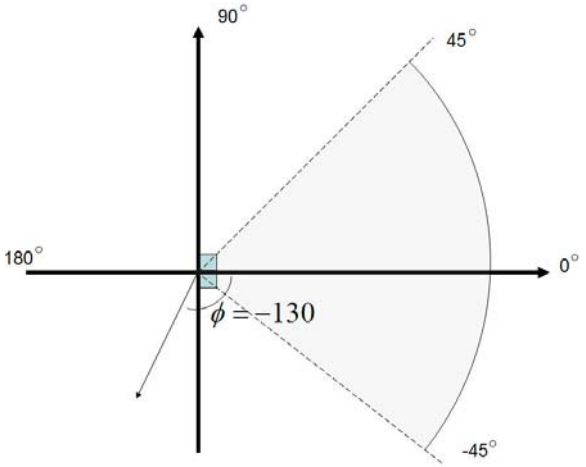
Now to adjust each line so that they all point to the right, if $\phi_A > 90°$ (Fig. 3.14a) then the recorded ϕ'_A is (Fig. 3.14b):

$$\phi'_A = 180° - \phi_A \tag{3.28}$$

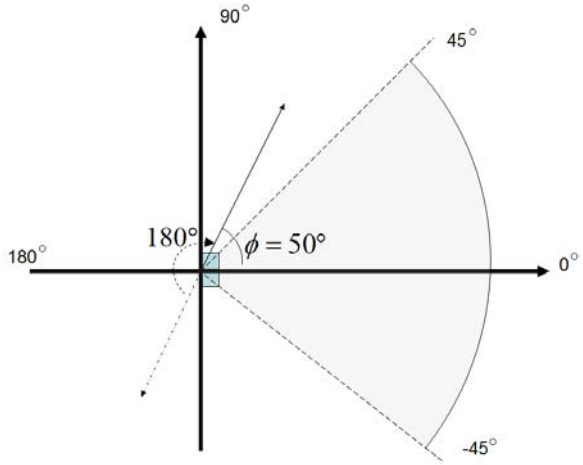
and if $\phi_A < -90°$ then:

$$\phi'_A = 180° + \phi_A \tag{3.29}$$

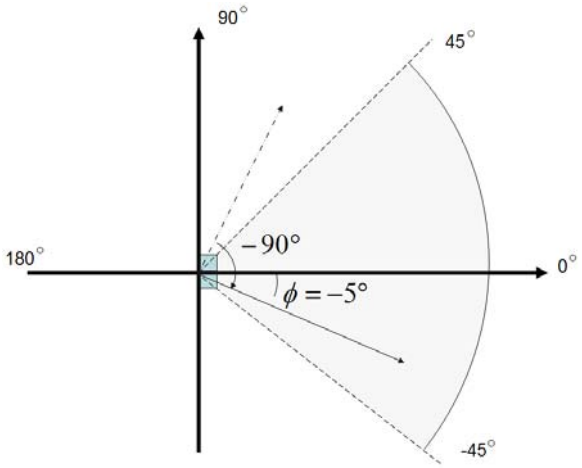
if neither of these conditions are met then $A' = A$.



(a) Angle adjustment when the angle is greater than 90°



(b) Angle adjustment when the angle is greater than 90°



(c) Angle adjust if the new angle is greater than 45°

Figure 3.14 Angle directions are adjusted to point as close as it can along the positive x axis.

Finally, if $\phi'_A > 45°$ then (Fig. 3.14c):

$$\phi'_A = 90° - \phi'_A \tag{3.30}$$

and if $\phi'_A < -45°$ then:

$$A' = -90° - \phi'_A \tag{3.31}$$

It is important to understand that this only gives us the s data set. This data, along with the center point of the exit locations, the m data set, is sent to the EM algorithm as described in Section 3.1.9.1 to find the actual direction of the ASR. The EM algorithm then uses the vectors and their positions by bringing ϕ'_A to the same orientation from the $x-$axis and moving the origins of A to its most probable location. The difference between the current ASR and the previous one is used as the error in localization to correct the robots pose. The algorithm to fill the s data set is shown in (Appendix A, Fig. A.7).

IV. Results and Analysis

Testing the implementation of the mapping system is accomplished modularly for each algorithm, and as a complete system. Results of these tests are shown and explained in the remainder of this chapter.

The first series of tests includes running the system using the Pioneer robot simulator which simulates the Pioneer P2-AT robot. The Pioneer P2-AT robot possesses a 400Mhz Pentium processor and 64MB of random access memory. Cell size in all results is $10_{cm} \times 10_{cm}$ by setting $\wp = 100$. Additionally, thresholds are set as shown in (Table 4.1).

Table 4.1 Thresholds and parameters used while mapping.

Symbol	Value	Description
ϵ	7	The value in a cell deemed occupied
τ	2	Closest allowed distance of a vertex
γ	4	Threshold for the Douglas-Peucker algorithm to determine edge angle cutoff
$\wp$	100	Scalar used to determine resolution of the map

Each of these thresholds were extensively tested, and these were found to produce the clearest map under the largest variety of mapping circumstances (large, small, cluttered, and uncluttered environments). Increasing τ, reduces the size of the final polyline map but will decrease the detail of the map at the same time. Changing the size of the scalar $\wp$ results in a inversely proportional change in the memory size of the map. Additionally, mapping was accomplished in real time with the robot traveling at approximately 0.5m/s.

The map in (Fig. 4.1a) shows a map created using the Pioneer robot simulator that includes 9 ASRs, and those rooms represented as a graph (Fig. 4.1b). In (Fig. 4.2) an example ASR before and after conversion to its simplified polygonal representation, mapped from rooms 5 and 6 of (Fig. 4.1a). The polygon requires less than 0.1KB, while the MHIMM used 42KB for representing the rooms. In (Fig.

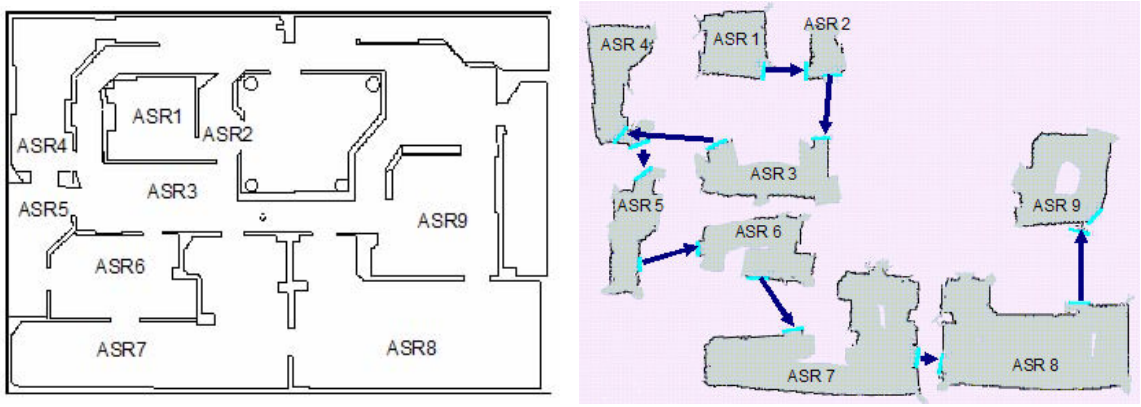
4.4) nine ASR's are mapped. When the rooms are put back together after mapping is complete, localization errors have caused overlapping; (Fig. 4.5) shows a similar map with less localization error.

In (Table 4.2) the respective file sizes of each representation of the map in (Fig. 4.6) is shown. Each stage of the process provides a substantially reduced data set, and (Fig. 4.6d) is only 1.06% the size of (Fig. 4.6b).

Table 4.2 Size(KB) differences of map representations

Map	**Size**
(Fig. 4.6a)	784
(Fig. 4.6b)	56
(Fig. 4.6d) and (Fig. 4.6f)	8

In (Fig. 4.1b) the separate ASRs are shown as a graph with their corresponding representations to the map in (Fig. 4.1a). This map was generated using the Pioneer robot simulator.

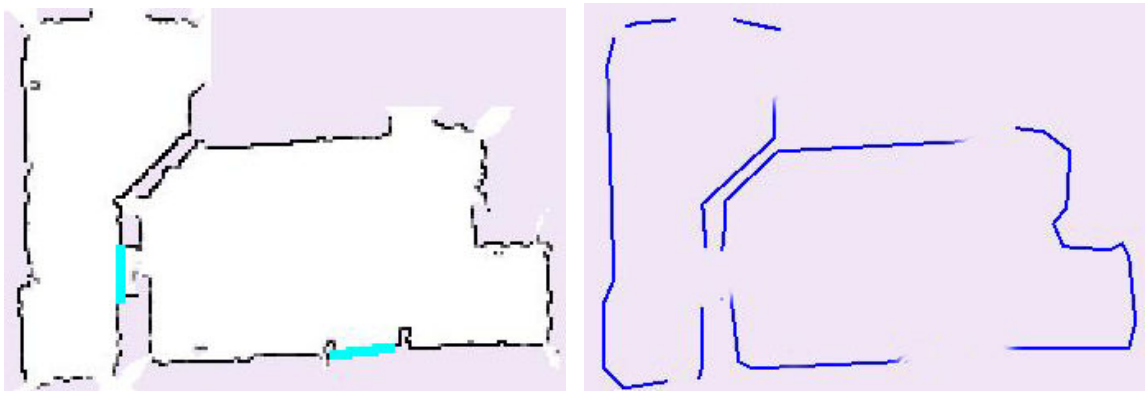


(a) (b)

Figure 4.1 ASR's Represented as a Graph. The actual map is shown in (a). The graph (b) shows how the rooms that were mapped translates to a graph.

Enlarging part of the map corresponding to rooms five and six of (Fig. 4.1a) in (Fig. 4.2a) is the array map with the rooms reconstructed to their proper location.

The final polyline map (without localization) of these two rooms is shown in (Fig. 4.2b).



(a) (b)

Figure 4.2 A 2 Room ASR Example showing the polygonal representation (b) of the original HIMM map (a). This figure is rooms 5 and 6 of (Fig. 4.1a).

In (Fig. 4.3) the complete process of mapping the nine rooms is shown without the localization step. As the robot moves from room to room the array map is created (left side of (Fig. 4.3)) and cleaned (right side of (Fig. 4.3)). But only the polyline information is maintained as the robot moves onto the next room. This process is accomplished real time while the robot maps the environment, without localizing.

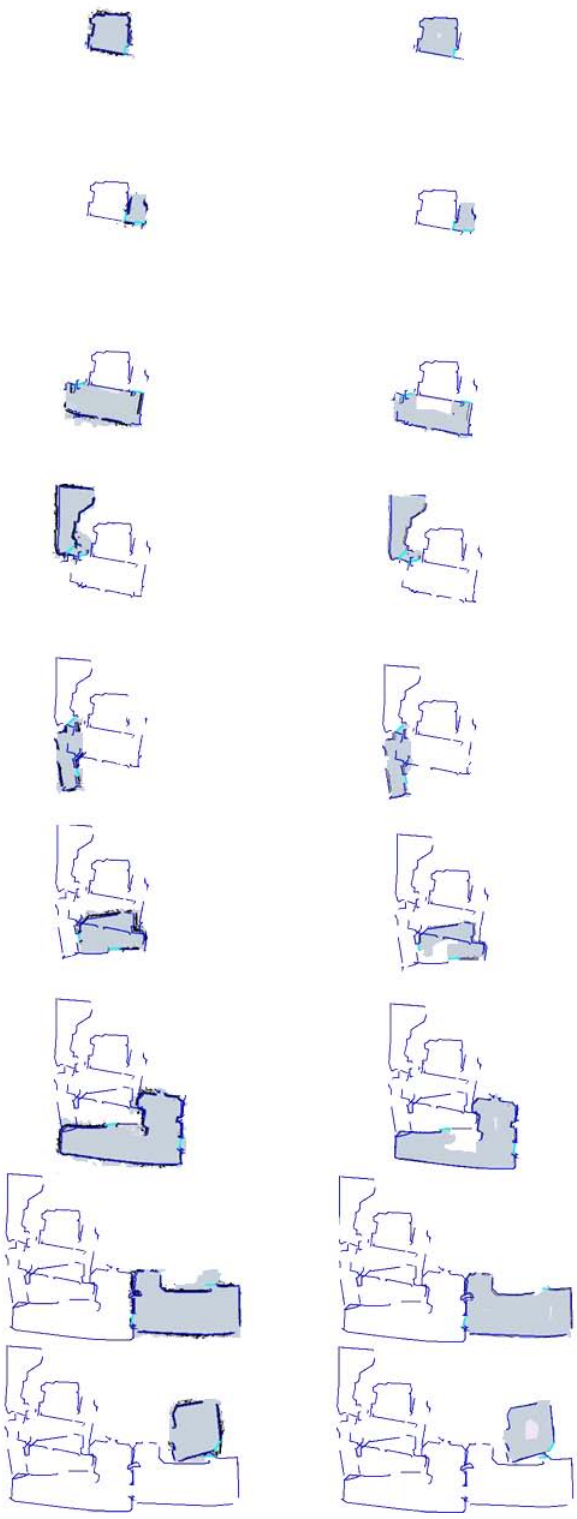


Figure 4.3 ASR Development

In (Fig. 4.4) is the completely reconstructed map from (Fig. 4.3), also showing the path of the robot.

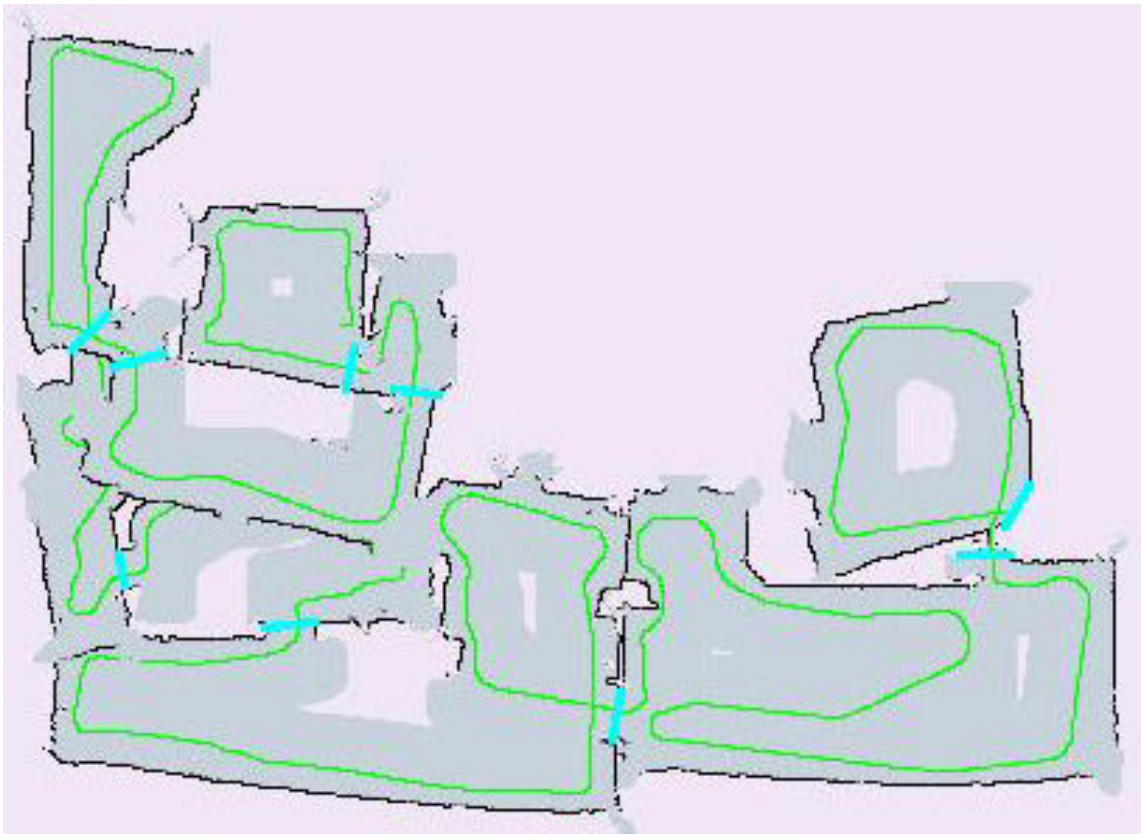
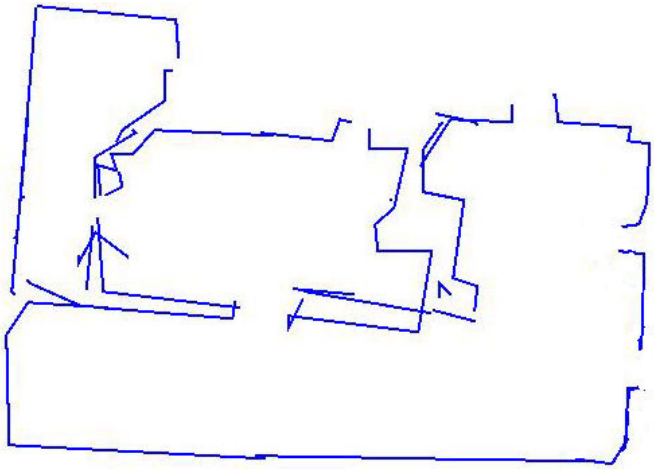
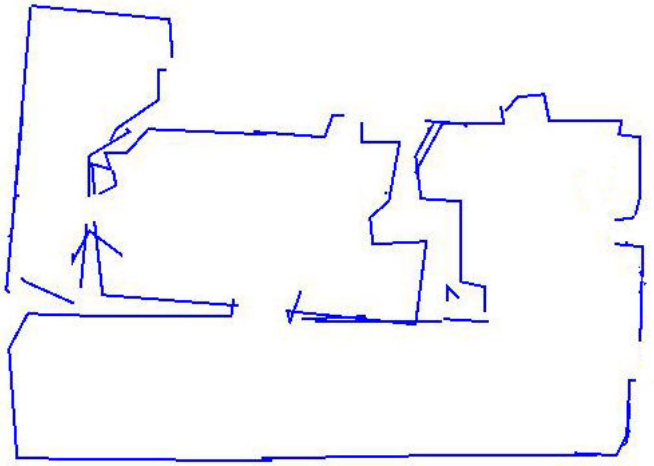


Figure 4.4 Nine ASR's Merged Together.

Applying the full SLAM algorithm, mapping only rooms five, six, and seven of (Fig. 4.4) gives the map in (Fig. 4.5b). The unlocalized map is shown in (Fig. 4.5a) for comparison. Using the system in a real world area produced better results then when using the Pioneer robot simulator. We show in (Fig. 4.6) a mapping of the first floor of the engineering building at the Air Force Institute of Technology, the black lined area of (Fig. 3.1), minus some offices with dimensions approximately $(255' \times 255')$. The robot is this figure is sent around the hallway one time with no overlap of areas mapped. On the left (Fig. 4.6a), is the grid representation of the map. On the right (Fig. 4.6b), the vertex map without simplification is illustrated



(a)



(b)

Figure 4.5 Three ASRs merged together showing little localization error to correct, (a) is the unlocalized polyline map and (b) is the localized polymap.

with a piece of the map expanded to show greater detail in (Fig. 4.6c). In (Fig. 4.6d) is the final simplified polyline map without localization with the top right corner blown up in (Fig. 4.6e). After localization (Fig. 4.6f) the map is almost corrected however some areas remain within the map that were unable to localize leaving slight imperfection. However, if there were overlap, i.e. the robot revisits the location, this localization error most likely would be rectified.

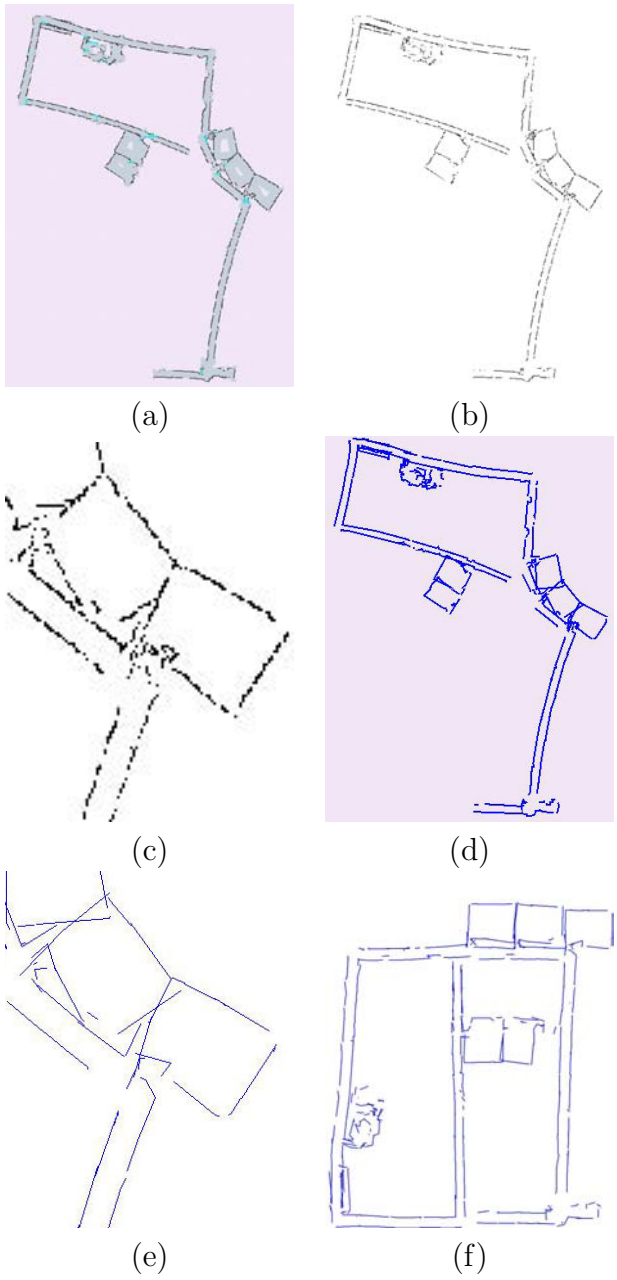


Figure 4.6 The polyline map of the first floor of the engineering building at the Air Force Institute of Technology. On the left (a), is the grid representation of the map, (b) is the vertex map without simplification, (c) shows a blowup of part of the map. In (d) is the final simplified polyline map without localization, (e) is a blowup of part of (d). The map after localization is shown in (f).

4.1 Small Complex Mapping Regions

Testing small complex spaces at first posed something of a problem which is solved by decreasing the scalar $\wp$. The scaling factor used in the larger maps resulted in incoherent smaller maps (Fig. 4.7). This map was made using the Pioneer robot simulator. A real world environment is shown later producing similar results.

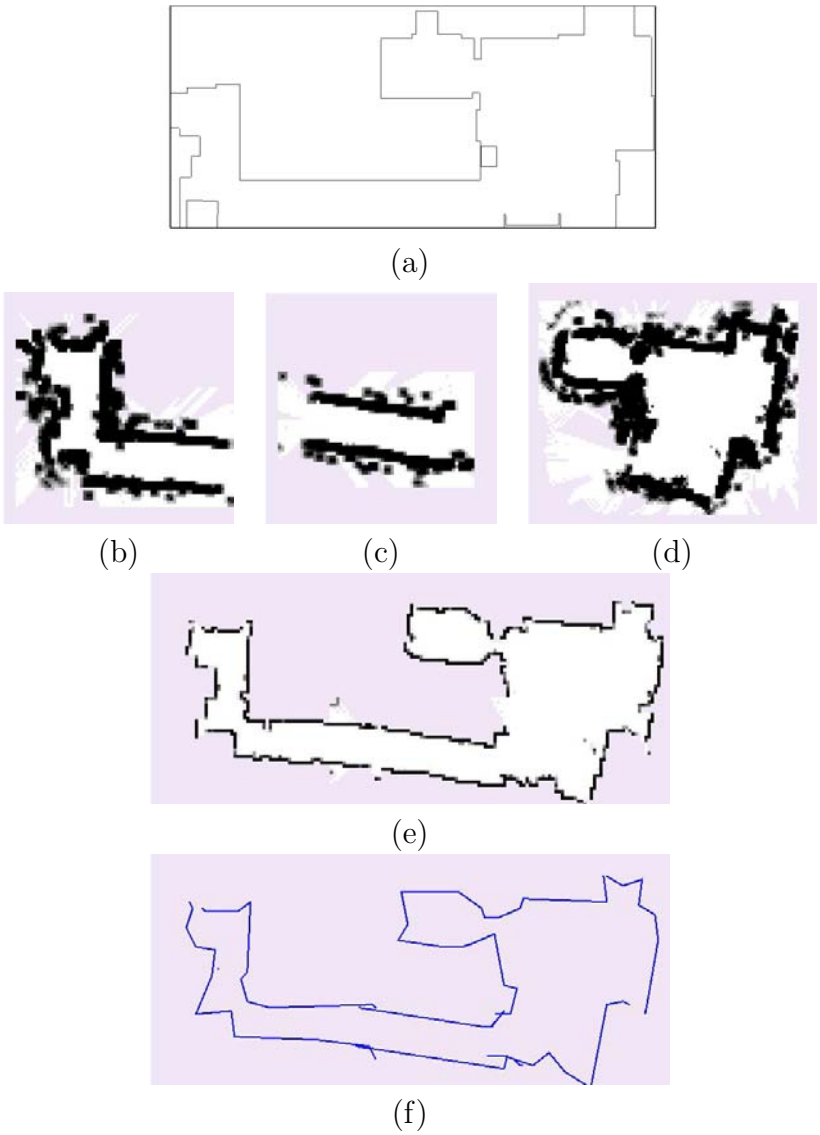


Figure 4.7 Stages of an example using the Pioneer robot simulator of a very small area being mapped. Area is approximately $20_{ft} \times 40_{ft}$. The scalar $\wp$ is set equal to 85.

However, reducing the scalar $\wp$ from 100 to 55 for more complex regions produces a better map (Fig. 4.8).

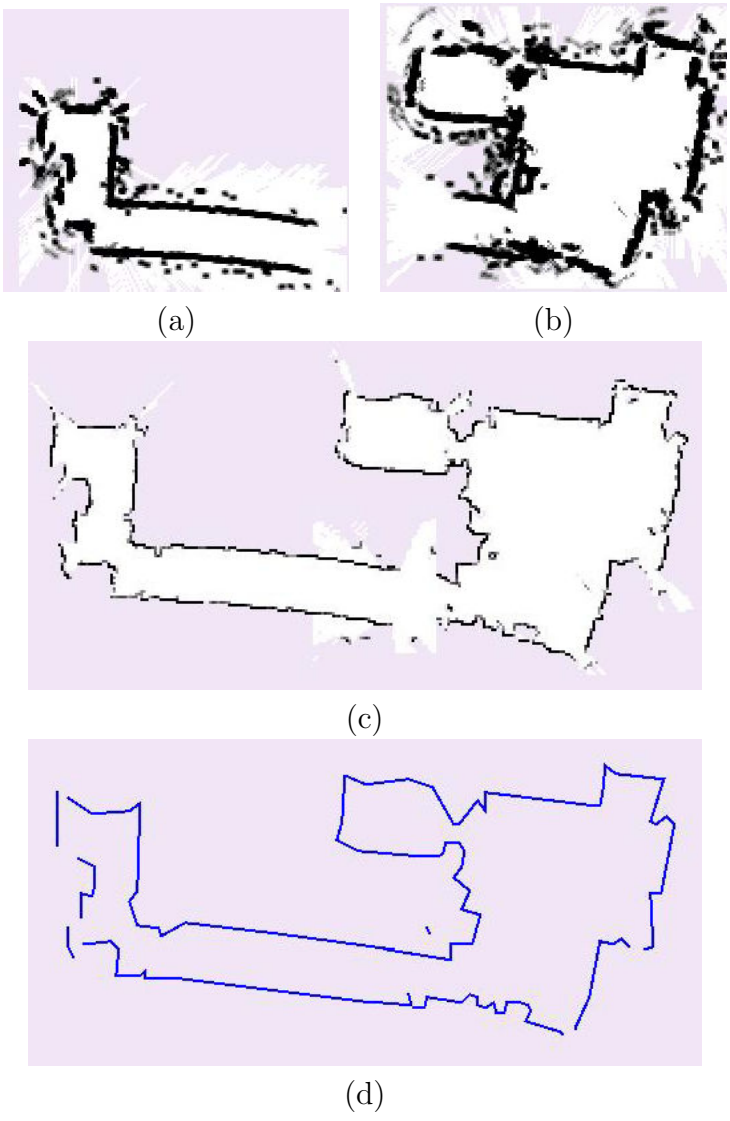


Figure 4.8 Stages of an example using the Pioneer robot simulator of a very small area being mapped. Area is approximately $20_{ft} \times 40_{ft}$. The scalar $\wp$ is set equal to 55.

If the region to be mapped is a large simple region, the reduced scaling factor makes little difference except perhaps an increased processing time as more information must be processed. Because of the small memory space needed using this

mapping system, the scale can be reduced very low even in a large, highly complex area with the potential to provide a surprisingly detailed map.

To test the small/messy type of environment in a real world environment, the robot was sent through part of another building (the AFIT robotics lab) which currently has items and furniture laying about almost randomly (Fig. 4.9). In this figure all the ASRs are shown as the raw array maps along with the robot path.

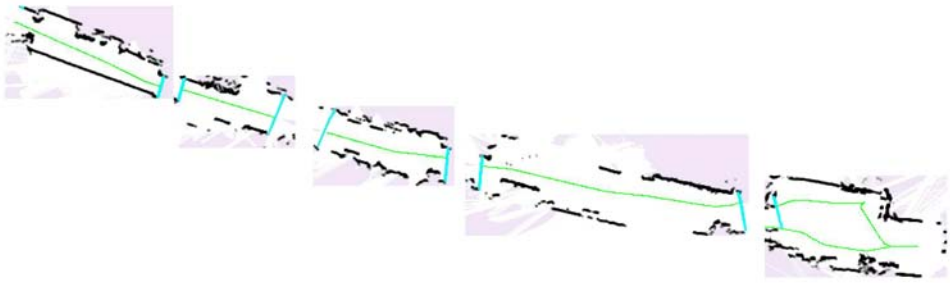


Figure 4.9 Small messy real world environment

(a)

(b)

Figure 4.10 Real world polyline map of a small cluttered region before (a) and after (b) localization.

4.2 Localization

Applying the localization algorithm in the hallway of the AFIT building, which is a relively simple environment but very large, produces the map in (Fig. 4.11). This environment represents the ideal arena for the SLAM algorithm implemented in this thesis.

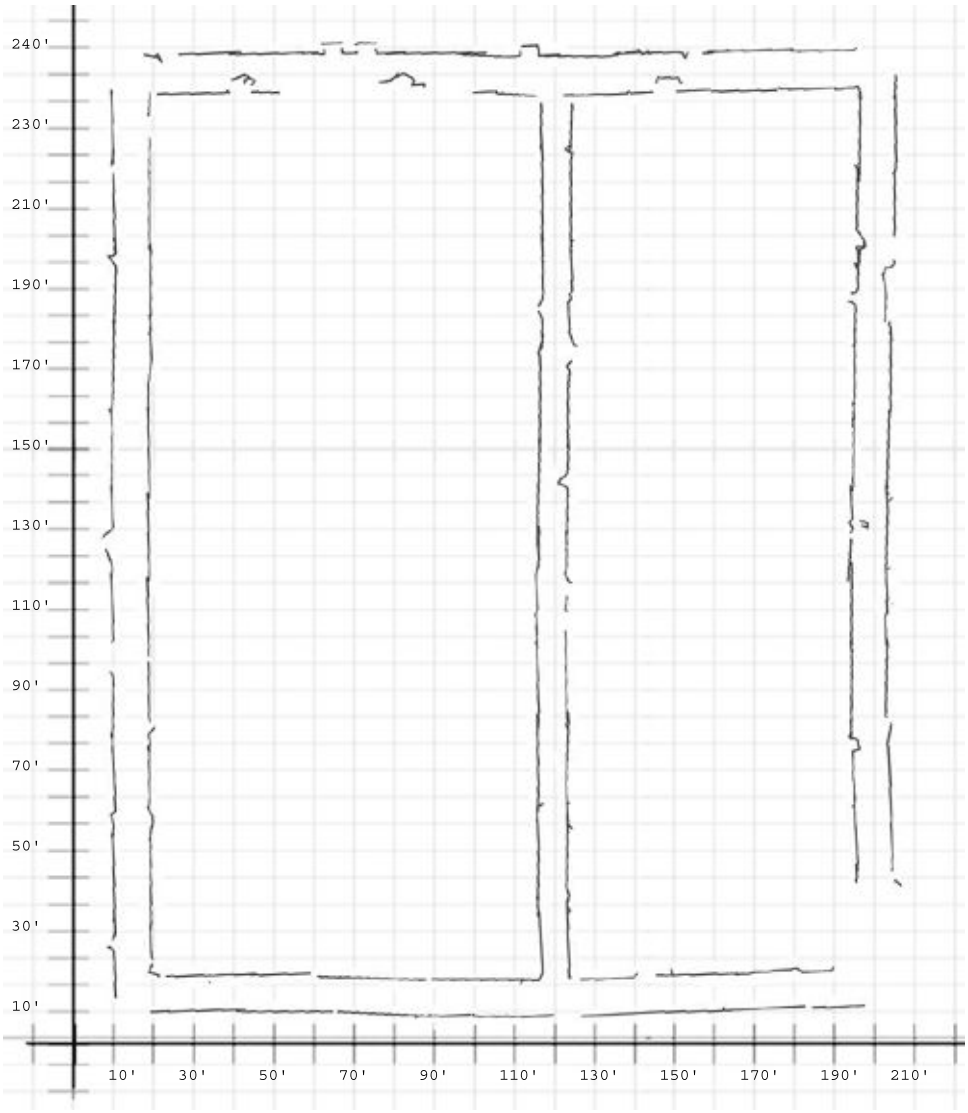


Figure 4.11 Localized map of the AFIT hallway.

Figure 4.11 is shown overlaid with the blueprint in (Fig. 3.1) in (Fig. 4.12).

Figure 4.12 Polyline map of the AFIT hallway overlaid with the blueprint.

The processing requirements to perform the mapping in (Fig. 4.11) does not incur any constraints, because the data set in which localization was performed on is so small. The total processing time in a batch mode with all of the data from a recorded robot took 38 seconds. A graph showing the processing time taken to process 1 to 43 of the 4m fixed sized ASRs in (Fig. 4.11) is shown in (Fig. 4.13). The processor time is calculated without included time required to traverse the environment and acquire sensor readings.

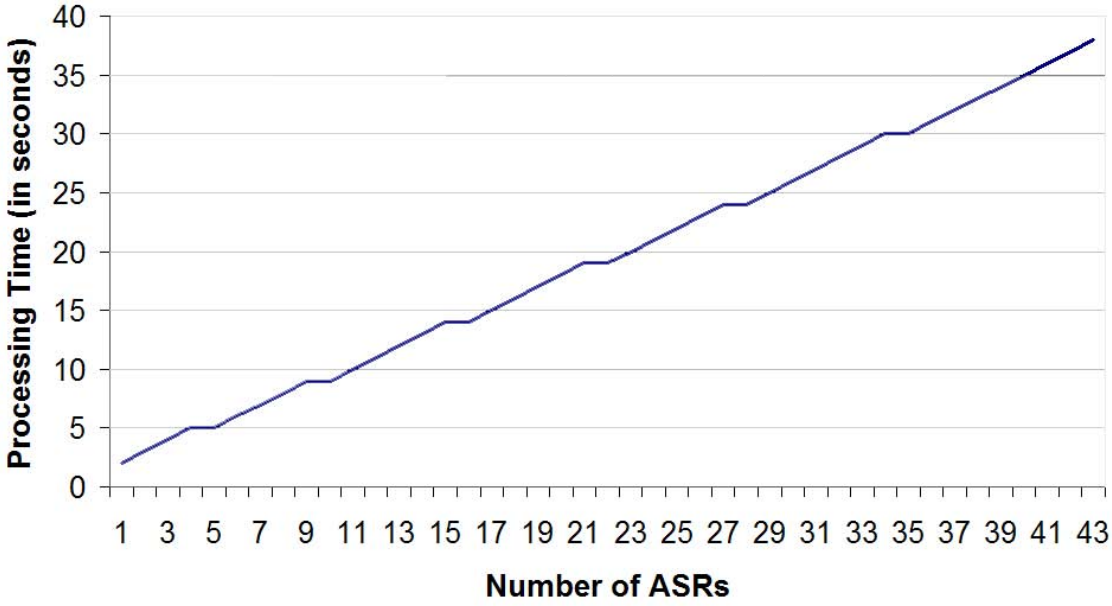


Figure 4.13 Processor time comparison required at which ASR in (Fig. 4.11)

The complete SLAM process performs in real-time. The robot is sent through the hall and as the robot progresses through the environment, ASRs are constructed and compared against the previous ASR to perform localization.

V. Future Work and Conclusion

While many methods exist for robots to map their environment [4][20][26][17][34], as the size of these environments increases, methods using occupancy grids start to approach maximum constraints [4][26][34]. The two-fold method (polylines and ASRs using sonars) [4][11][17] combines the best of two robot mapping philosophies. Our system makes it possible for a robot armed only with sonars to map an arbitrarily large area. It is size scalable, with a non-increasing computational time required to map and a linearly increasing computational time required to reconstruct the ASRs after completion of mapping, that is, create a single global map using the individual ASRs. Additionally, the reduced data set that makes up the polylines are used to perform localization using the Expectation Maximization (EM) algorithm.

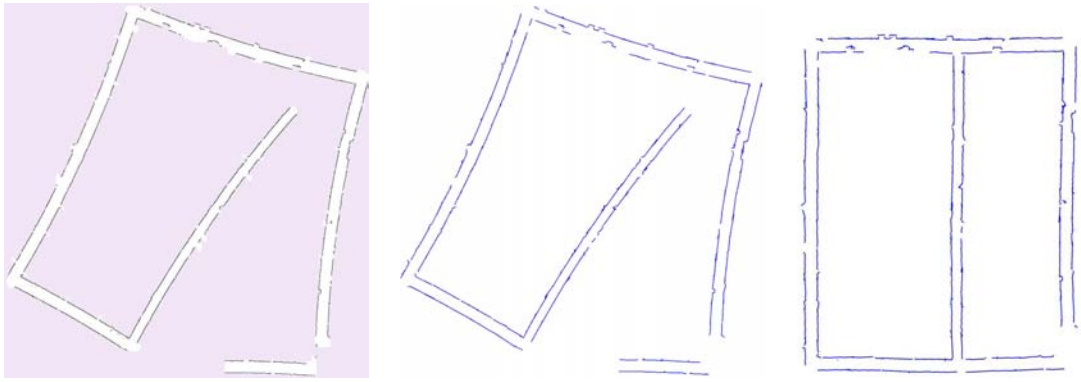
The objectives of this thesis was to:

1. Implement and test a fast histogram mapping algorithm which was done using a modification to the HIMM mapping technique.
2. Develop a polygonal and topological representation from the occupancy grid map. This was accomplished using a new technique of filtering the HIMM map and converting the data to a vertex list and applying the Douglas Peuker algorithm to simplify the list.
3. Implement and test an Expectation Maximization algorithm to perform localization corrections of the cognitive map. This was also successfully accomplished fitting polygonal data into a representation which could be maximized with the EM algorithm.

All objectives were successfully completed as was the goal for providing a fast (using HIMM technique), accurate (EM and linear fitting) method using probability theory, to quickly map a large area. Additionally, the map is stored in a small amount

of memory (polygonal ASR representation) where if needed, it could be transmitted quickly to another device.

The cognitive mapping process allows robots to easily implement corrections to localization errors using the simplified ASRs instead of grid data. In figure 5.1, a before and after map is shown with a memory footprint of the map. Additionally, the ASR is reduced to a fixed size of 4m, allowing the robot to perform iterative localization using an expectation maximization algorithm similar to Thrun, et al. [36], matching the current array window of the robot with the previous ASRs to determine the more probable current pose. This idea is taken from work developed by Schultz and William [33], where range sensor data is divided up into several time slices and comparing them to determine an estimate of pose error. Our idea is to extend this technique to ASRs and fix the size of the ASR to some small value then compare them against each other in a similar fashion. This technique produces good results. The map in Figure 5.1b is the polygonal representation of the map in Figure 5.1a. After localization adjusting rotation sensor error among room ASRs, yields Figure 5.1c (this is the same map shown in (Fig. 4.11)).



(a) (b) (c)

Figure 5.1 The array map of the first floor of the engineering building at the Air Force Institute of Technology is shown in Figure (a). Figure (b) shows the polyline map with localization errors uncorrected, figure (c) shows the polyline map after rotational localization errors are fixed.

5.0.1 Extensions. The technique used in this thesis affords the opportunity of several extensions. While sonars are very common in robot mapping systems, they are an older *active* method of sensing. The term *active* describes a sensor that needs to send some sort of signal to take a measurement, unlike a *passive* sensor which requires no emitted signal to gain a measurement. Replacing or enhancing range data with a passive camera system could produce a cleaner, more accurate map while at the same time not generate any detectable signal.

Using ASRs as a means of representing the map lends itself to multi-robot mapping. It is an easy extension to use multiple robots, each mapping separate ASRs at the same time and at some point communicating the maps to each other and merging them to one comprehensive map. This would require some additional calculations to determine where each robot is with respect to each other but not much more.

This implementation places a low emphasis in exit detection. Many possible configurations of exits exist including:

1. A room into another room through a doorway (Fig. 5.2a)
2. Room to a hallway that leads to yet another room (Fig. 5.2b)
3. From the middle of a room to a hallway to the side of another room (Fig. 5.2c)
4. Side of a room to a hallway to the side of another room (Fig. 5.2d)
5. Same but no hallway (Fig. 5.2e)
6. Partition in one room (Fig. 5.2f)

These are just a few of the many possible configurations. Of this list, only (Fig. 5.2a), (Fig. 5.2e), and (Fig. 5.2f) can be detected with the algorithm used in this thesis. An obvious extension is to include a more sophisticated exit detection system that can handle a wider variety of exit configurations.

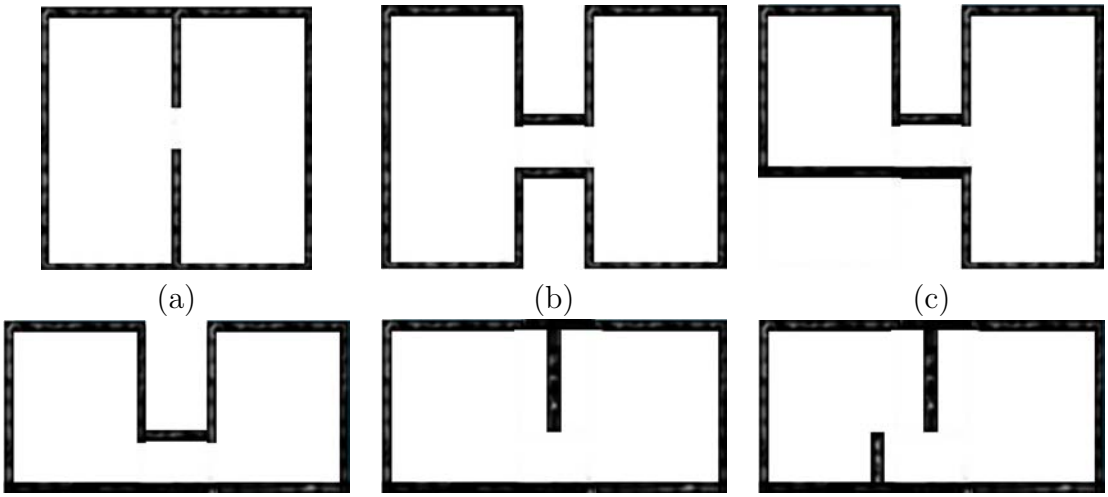


Figure 5.2 Various types of exits the robot may typically encounter.

While this system will map an environment that changes slightly with time without a serious degradation of the map it is not designed specifically to handle dynamic environments. Another enhancement to the system is the addition of an agent responsible for identifying changes to the environment and sequentially registering those changes to the appropriate ASR.

Appendix A. Algorithms

A.1 Mapping Algorithms

Section 2.2 and Section 3.1.3 gives the definitions shown in (Table A.1) that are used in this section to develop a general algorithm for grid mapping.

Table A.1 Symbol key for the mapping algorithms.

Symbol	**Definition**
$\wp$	Scalar used to determine cell size.
C	2 dimensional matrix that represents the world.
m	Max distance divided by $\wp$ expected in the x direction.
n	Max distance expected to travel in the y direction divided by $\wp$.
h	Robots direction.
sh_i	Each sonar's angle relative to the robot. $i = \{1, 2, 3, ...number\ of\ sonars\}$
h_i	The global angle of each sonar. $h_i = h + sh_i$.
sox_i	Sonar$_i$n x offset from the center of the robot.
soy_i	Sonar$_i$n y offset from the center of the robot.

Map(C)
Loop
$so \leftarrow$ New sonar readings
$h \leftarrow GetPoseH(robot)$ //Vehicle direction
$x_0 \leftarrow GetPoseX(robot)$ //Vehicle center point x coordinate
$y_0 \leftarrow GetPoseY(robot)$ //Vehicle center point y coordinate
For ($i = 1$ to number of sonars)
$x_i \leftarrow \frac{r_i \times \cos(h_i) + (sox_i \cos(h) - soy_i \sin(h))}{\wp}$
$y_i \leftarrow \frac{r_i \times \sin(h_i) + (sox_i \sin(h) - soy_i \cos(h))}{\wp}$
Update(C_{x_i, y_i})
End For
End Loop

Figure A.1 General Occupancy Grid Mapping Algorithm

Map(C)
Loop
$so \leftarrow$ New sonar readings
$h \leftarrow GetPoseH(robot)$ *//Vehicle direction*
$x_0 \leftarrow GetPoseX(robot)$ *//Vehicle center point x coordinate*
$y_0 \leftarrow GetPoseY(robot)$ *//Vehicle center point y coordinate*
For ($i = 1$ to number of sonars)
$x_i \leftarrow \frac{r_i \times \cos(h_i) + (sox_i \cos(h) - soy_i \sin(h))}{\wp}$
$y_i \leftarrow \frac{r_i \times \sin(h_i) + (sox_i \sin(h) - soy_i \cos(h))}{\wp}$
If ($dist(x_i, y_i, x_0, y_0) < w$) **Than** Update$(C_{x_i,y_i})$
End For
End Loop

Update$(C_{tx,ty})$
$C_{tx,ty} = C_{tx,ty} + 1.5$
For ($l = -1$ **to** 1)
For ($m = -1$ **to** 1)
$C_{tx+m,ty+l} = C_{tx+m,ty+l} + 1.5$
If ($C_{tx+m,ty+l} > 15$) than ($C_{tx+m,ty+l} = 15$)
End For
End For
//decrease all cells by 1 in the line from (tx, ty) *to* (x_0, y_0)*. Can drop to 0.0. Bresenham's line algorithm [6] is used to follow the line.*
LineDecrease(tx, ty, x_0, y_0)

Figure A.2 Histogram In Motion Mapping Algorithm

Map(C)
Loop
$so \leftarrow$ New sonar readings
$h \leftarrow GetPoseH(robot)$ *//Vehicle direction*
$x_0 \leftarrow GetPoseX(robot)$ *//Vehicle center point x coordinate*
$y_0 \leftarrow GetPoseY(robot)$ *//Vehicle center point y coordinate*
For ($i = 1$ to number of sonars)
$x_i \leftarrow \frac{r_i \times \cos(h_i) + (sox_i \cos(h) - soy_i \sin(h))}{\wp}$
$y_i \leftarrow \frac{r_i \times \sin(h_i) + (sox_i \sin(h) - soy_i \cos(h))}{\wp}$
If $(dist(x_i, y_i, x_0, y_0) < w)$ **Than** Update(C, x_i, y_i, x_0, y_0)
Else
$x_i \leftarrow \frac{w \times \cos(h_i) + (sox_i \cos(h) - soy_i \sin(h))}{\wp}$
$y_i \leftarrow \frac{w \times \sin(h_i) + (sox_i \sin(h) - soy_i \cos(h))}{\wp}$
//decrease all cells by 1 in the line from (x_i, x_i) *to* (x_0, y_0)*. Can drop to -3.*
LineDecrease(C, t_i, t_i, x_0, y_0)
End For
End Loop

Update(C, tx, ty, x_0, y_0)
$C_{t_0,t_0} \leftarrow -100$
$C_{tx,ty} \leftarrow C_{tx,ty} + 1.5$
For ($l = -1$ **to** 1)
For ($m = -1$ **to** 1)
$C_{tx+m,ty+l} \leftarrow C_{tx+m,ty+l} + 1.5$
If $(C_{tx+m,ty+l} > 15)$ **than** $C_{tx+m,ty+l} \leftarrow 15$
End For
End For
LineDecrease(C, tx, ty, x_0, y_0)
//decrease all cells by 1 in the line from (tx, ty) *to* (x_0, y_0)*.*
//Allowed to drop to -3.
//Bresenham's line algorithm [6] is used to follow the line.

Figure A.3 Modified Histogram In Motion Mapping Algorithm

A.2 Simplification Algorithms

The process of simplifying the list of vertices V extracted from the map begins by eliminating clusters of closely grouped points. A tolerance value τ is defined such that for any point V_i, if $\Delta(V_i, V_{i+m}) < \tau$ for $m = \{i, i+1, i+2, i+3, ...n\}$, then V_{i+m} is removed from V. When $\Delta(V_i, V_{i+m}) > \tau$ then i is set $i \leftarrow i + m$.

VertexSimplify(V)
ϵ =2 /*The bigger ϵ the less accurate the map will be 2 seems to work good.*/
For (from 0 **to** $|V| - 2$ as i)
For (from i **to** $|V|$ as m)
$d \leftarrow \Delta(V_i, V_{i+m})$.
If $(d < \epsilon)$ $V = V - \{V_{i+m}\}$ /*that is, V_{i+m} is removed from V*/
If $d \geq \epsilon$ **Then** Break

Figure A.4 VertexSimplify Algorithm

Let η be a specified tolerance for the smallest angel algorithm should recurse on where if the angle is greater than η, then the approximation is acceptable and the endpoints are retained. The simplify function is initialized with $W_0 = W_n$ =*true*, where W_0 is the first vertex and W_n is the last vertex. Let Wo be the set of output vertices.

PolySimplify()
Simplify$(\eta, W, 0, n, mark)$
For (0 to $|W|$ as i)
If $(mark_i)$
then add it to the output set Wo
$Wo_{next} \leftarrow W_i$
Simplify(Float $\eta, Set\ L, Int\ j, Int\ k, Set\ mark$)
L is the original set of points mark is the vector used to identify points to keep. j and k are used to keep track of the current and last point.
If $(k < j)$ */*Nothing to simplify*/*
Return
Segment S
$S_{jk} \leftarrow$ *Segment from* L_j *to* L_k
$max_{distance} \leftarrow 0$
$max_i \leftarrow j$
Let u be the segment direction vector
$u_x \leftarrow S_{b_x} - S_{a_x}$
$u_y \leftarrow S_{b_y} - S_{a_y}$
$c_u \leftarrow (u \bullet u)$
P_b is the base of perpendicular from v_i to S
b, cw, d_{v2} */*$d_{v2} \leftarrow$ dist(v_i , S^2)*/*
For $(j+1$ **to** k **as** $i)$
$d_{v2} \leftarrow$distance squared $(L_i,\ P_b)$
If $(d_{v2} \leq max_d)$ *continue*
Else
$max_i \leftarrow i$
$max_d \leftarrow d_{v2}$
If $(max_d > \eta)$
*/*Split the polyline at the farthest vertex from S*/*
$mark_{max_i} \leftarrow$ *true*
simplify$(\eta,\ L,\ j,\ max_i,\ mark)$
simplify$(\eta,\ L,\ max_i,\ k,\ mark)$
For(1 *to* $k-1$ as i)
If$((L_{(i-1)_x} < 0) \cup (L_{i_x} < 0) \cup (L_{(i+1)_x} < 0))\ mark_i \leftarrow$ *true*

Figure A.5 Douglas-Peucker Edge Simplification Algorithm

A.3 The EM Algorithm

We have a set of parameters $\theta_k = (\pi_k, \phi_k)$ that defines a probability distribution function of the robot's location, where $k = s, m$; s being the abstracted sonar data detailed in Section 3.1.10 and m the motor data of the probability distribution function. All other variables and functions are defined as follows:

- $k = \{s, m\}$, $x = \{x_1, x_2, x_3, ...x_N\}$ of s or m and $\phi_k = \{\mu_k, \sigma_k\}$ as defined in the previous section.
- Mean(X) calculates the mean of the set of X, μ_X

  Returns: $\frac{\sum_{i=1}^{|X|} X_i}{|X|}$
- StdDev(X) calculates the standard deviation of the set X, σ_X

  Returns: $\sqrt{\frac{\sum_{i=1}^{|X|}(X_i - Mean(X))}{|X|-1}}$
- Var(X) calculates the variance of the set X, σ_X^2

  Returns: $\frac{\sum_{i=1}^{|X|}(X_i - Mean(X))}{|X|-1}$
- Gauss(x_i, μ, σ) is the function of $p(x_i|\mu, \sigma^2)$

  Returns: $\frac{\exp(-\frac{(x_i-\mu)^2}{2\sigma^2})}{\sqrt{2\pi}\sigma}$
- SumGaussk($x_i, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m$) is the function of $\sum_k p(x_i|\mu_k, \sigma_k^2)$

  Returns: π_s Guess(x_i, μ_s, σ_s) + π_m Guess(x_i, μ_m, σ_m)
- SumGausskTimesLog($x_i, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m$) is the function of $\sum_k \left(< q_i^k > \log(\pi_k p(x_i|\phi_k))\right)$

  Returns: q($x_i, \pi_s, \pi'_s, \mu'_s, \sigma'_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m$) $\times$ log(π_sGuess(x_i, μ_s, σ_s))+ q($x_i, \pi_s, \pi'_m, \mu'_m, \sigma'_m, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m$) $\times$ log(π_mGuess(x_i, μ_m, σ_m)
- q($x_i, \pi_s, \pi'_k, \mu'_k, \sigma'_k, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m$) is the function of $< q_i^k >$

  Returns: π'_kGuess(x_i, μ'_k, σ'_k)/(SumGaussk($N, x_i, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m$))

- sum$< q >(k, \mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_k, \mu'_k, \sigma'_k)$

  For ($i = 1$ to $|\mathbf{x}|$)

  sum = sum + $q(k, x_i, \pi_s, \pi'_k, \mu'_k, \sigma'_k, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m)$

  End For

  Return: sum

- sumqXi$(k, \mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_k, \mu'_k, \sigma'_k)$

  For ($i = 1$ to $|\mathbf{x}|$)

  sum = sum + $x_i \times q(k, x_i, \pi_s, \pi'_k, \mu'_k, \sigma'_k, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m)$

  End For

  Return: sum

- sumqXiMinusMu$(k, \mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_k, \mu'_k, \sigma'_k, \widehat{\mu_k})$

  For ($i = 1$ to $|\mathbf{x}|$)

  sum = sum + $q(k, x_i, \pi_s, \pi'_k, \mu'_k, \sigma'_k, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m) \times (x_i - \widehat{(}\mu_k))^2$

  End For

  Return: sum

PerformEM$(\theta, k, \mathbf{x})$
Initialize π_k *to some random numbers*
While (error$> \epsilon$)
Perform E-Step
  For $(N = 1$ to $|\mathbf{x}|)$
    $x_N \leftarrow$ E-Step$(\theta, \theta', \mathbf{x}, N)$
  End For
Perform M-Step
double SumqS$\leftarrow$sumq$(\mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_s, \mu'_s, \sigma'_s)$
double SumqM$\leftarrow$sumq$(\mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_m, \mu'_m, \sigma'_m)$
  $\widehat{\pi_s} \leftarrow$SumqS$/|\mathbf{x}|$
  $\widehat{\pi_m} \leftarrow$ SumqM$/|\mathbf{x}|$
  $\widehat{\mu_s} \leftarrow$ sumqXi$(\mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_s, \mu'_s, \sigma'_s)$/SumqS
  $\widehat{\mu_m} \leftarrow$ sumqXi$(\mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_m, \mu'_m, \sigma'_m)$/SumqM
  $\widehat{\sigma_s}^2 \leftarrow$sumqXiMinusMu$(\mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_s, \mu'_s, \sigma'_s, \widehat{\mu_s})$/SumqS
  $\widehat{\sigma_m}^2 \leftarrow$sumqXiMinusMu$(\mathbf{x}, \pi_s, \pi_m, \mu_s, \mu_m, \sigma_s, \sigma_m, \pi'_m, \mu'_m, \sigma'_m, \widehat{\mu_s})$/SumqM
  $\pi'_s \leftarrow \pi_s$
  $\pi_s \leftarrow \widehat{\pi_s}$
  $\mu'_s \leftarrow \mu_s$
  $\mu_s \leftarrow \widehat{\mu_s}$
  $\sigma'^2_s \leftarrow \sigma^2_s$
  $\sigma^2_s \leftarrow \widehat{\sigma^2_s}$
  $\pi'_m \leftarrow \pi_m$
  $\pi_m \leftarrow \widehat{\pi_m}$
  $\mu'_m \leftarrow \mu_m$
  $\mu_m \leftarrow \widehat{\mu_m}$
  $\sigma'^2_m \leftarrow \sigma^2_m$
  $\sigma^2_m \leftarrow \widehat{\sigma^2_m}$
  $error \leftarrow |\mu_s - \mu'_s| + |\mu_m - \mu'_m|$
End While Loop

Figure A.6 Expectation Maximization Algorithm

A.4 Algorithm to Convert Vectors to Angles

To calculate s from each ASR, a direction for each line within an ASR is calculated to obtain an ASR direction. Let A denote the current vector and B the vector that comes before A. Let U be the normal vector along the positive x axis of the cartesian map represented by the current ASR and V be the complete vertex list.

Gets(Vertex List V)
  For($i = 1$ **To** $|V|$)
    $A = V_i$
    $B = V_{i-1}$
    $A_x \leftarrow A_x - B_x$
    $A_y \leftarrow A_y - B_y$
    $\phi = \cos^{-1}\left(\frac{A_x}{|A|}\right)$
    $\phi \leftarrow \frac{\phi}{\pi} \times 180$
    If $(A_x < 0)$ $\phi \leftarrow -\phi$
    If $(\phi > 90)$ $\phi \leftarrow 180 - \phi$
    If $(\phi < -90)$ $\phi \leftarrow 180 + \phi$
    If $(\phi > 45)$ $\phi \leftarrow 90 - \phi$
    If $(\phi < -45)$ $\phi \leftarrow 90 + \phi$
    $s \leftarrow s \cup \{\phi\}$
  End For
  Return s

Figure A.7 Calculating a value for sensor data from the abstracted map.

Bibliography

1. L. E. Baum, T. Petrie, G. Soules, and N. Weiss, *A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains.* Annual Math Stat. Vol. 41, pp. 164-171, 1970.

2. R. C. Bolles and P. Horaud, *3DPO: A Three-Dimensional Part Orientation System.* International Journal of Robotics Research, Vol. 5(2), pp. 326, 1986.

3. J. Borenstein, B. Everett, and L. Feng, *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, Ltd. Wellesley, MA, 1996.

4. J. Borenstein and Y. Koren, *Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance.* IEEE Journal of Robotics and Automation, Vol. 7, No. 4, pp. 535-539, 1991.

5. J. Borenstein and Y. Koren, *The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots.* IEEE Journal of Robotics and Automation, Vol. 7, No. 4, pp. 278-288, 1991.

6. N. Cossitt, *Introduction to Bresenham's Line Algorithm Using the SBIT Instruction.* Series 32000 Graphics Note 5, National Semiconductor Application Note 524, 1988.

7. F. Dellaert, W. Burgard, F. Dieter, and S. Thrun, *Using the Condensation Algorithm for Robust, Vision-Based Mobile Robot Localization.* Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR99), Fort Collins, CO, June 1999.

8. A. Dempster, N. Laird, and D. Rubin, *Maximum likelihood from Incomplete Data via the EM Algorithm.* Journal of the Royal Statistical Society, Series B, Vol. 39(1), pp. 1-38, 1977.

9. D. Douglas and T. Peucker, *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature.* The Canadian Cartographer, Vol. 10(2), pp. 112-122, 1973.

10. O. D. Faugeras and M. Hebert, *The Representation Recognition, and Locating of 3D Objects.* Int. J. of Robotics Research, Vol. 5(3), pp. 27-52, 1986.

11. H. Gonzanos and J. Latombe, *Navigation Strategies for Exploring Indoor Environments.* International Journal of Robotics Research, 2001.

12. W. E. L. Grimson, *Object Recognition by Computer: The Role of Geometric Constraints.* The Massachusetts Institute of Technology Press, Cambridge, Massachusetts, 1990.

13. R. Hill, C. Han, and M. Lent, *Applying Perceptually Driven Cognitive Mapping to Virtual Urban Environments.* AI magazine, Vol. 23, 2002.

14. P. Hough, *A Method and Means for Recognizing Complex Patterns.* U.S. Patent, Number 3,069,654, 1962.

15. M. E. Jefferies and W. K. Yeap, *Neural Network Approaches to Cognitive Mapping.* Artificial Neural Networks and Expert Systems, 1995. In Proceedings of the Second New Zealand International Two-Stream Conference, pp. 75-78, November 1995.

16. M. E. Jefferies and W. K. Yeap, *The Utility of Global Representations in a Cognitive Map.* In Proceedings of the 2001 Conference on Spatial Information Theory, 2001.

17. M. E. Jefferies, W. K. Yeap, L. Smith, and D. Fergusen, *Building a Map for Robot Navigation Using a Theory of Cognitive Maps.* In Proceedings of the IASTED International Conference on Artificial Intelligence and Applications, Marbella, Spain, 2001.

18. P. Jensfelt, *Approaches to Mobile Robot Localization in Indoor Environments.* Doctoral Thesis, Royal Institute of Technology, Department of Signals, Sensors and Systems, 2001.

19. P. Jensfelt, O. Wijk, D. J. Austin, and M. Andersson, *Experiments on a Augmenting Condensation for Mobile Robot Localization.* In proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA, April 2000.

20. D. Kortenkamp, *Cognitive Maps for Mobile Robots: A Representation for Mapping and Navigation.* PhD Thesis, University of Michigan, 1993.

21. B. Kuipers and Y. Byun, *A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations.* Qualitative Reasoning Group at the Artificial Intelligence Laboratory, Research of the Qualitative Reasoning Group is supported in part by NSF grants IRI-8602665, IRI-8905494, and IRI-8904454, and by NASA grants NAG 2-507 and NAG 9-200.

22. G. B. Lamont, Course Material, CSCE 686 Advanced Algorithm Design, Air Force Institute of Technology, 2003.

23. D. G. Lowe, *Object Recognition from Local Scale Invariant Features.* In Proceedings of the Seventh International Conference on Computer Vision (ICCV99), pp. 11501157, Kerkyra, Greece, September 1999.

24. J. Malik, Course Material, CS 294-2 Grouping and Recognition, University of California at Berkeley, 1999.

25. J. S. Milton and J. C. Arnold, "Introduction to Probability and Statistics: Principles and Applications for Engineering and Computing Sciences". McGraw-Hill Primis Custom Publishing, 2002.

26. H. Moravec, *Sensor Fusion in Certainty Grids for Mobile Robots.* In Sensor Devices and Systems for Robotics, Springer-Verlag, Nato ASI Series, pp. 253-276. 1989.

27. R. R. Murphy, *Dempster-Shafer Theory for Sensor Fusion in Autonomous Mobile Robots.* IEEE Transactions on Robotics and Automation, 14(2):197-206, 1998.

28. J. Neira and J. D. Tardos, *Data Association in Stochastic Mapping Using the Joint Compatibility Test.* IEEE Transactions on Robotics and Automation, Vol. 17, No. 6, pp. 890-897, December 2001.

29. J. Neira, J. D. Tardos, and J. A. Castellanos, *Linear Time Vehicle Relocation in SLAM.* Universidad de Zaragoza.

30. G. L. Peterson, Course Material, CSCE 723 Advanced Topics in Artificial Intelligence, Air Force Institute of Technology, 2003.

31. A. J. Pettofrezzo, "Matrices and Transformations". Dover Publications, Inc. New York, 1978.

32. S. Se, D. Lowe, and J. Little, *Global Localization using Distinctive Visual Features* . International Conference on Intelligent Robots and Systems, 2002.

33. A. C. Schultz and A. William, *Continuous Localization Using Evidence Grids.* In proc. of the IEEE International Conference on Robotics and Automation, Leuven, Belgium, pp.2833-2839, May 16-21, 1998.

34. S. Thrun, *Probabilistic Algorithms in Robotics.* AI Magazine, Vol. 21: pp. 93-109, 2000.

35. S. Thrun, *Robotic Mapping: A Survey.* In G. Lakemeyer and B. Neberl, editors, Exploring Artificial Intelligence in the New Millennium. Morgan Kaufmann, 2002.

36. S. Thrun, W. Burgard, and D. Fox, *A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots.* Machine Learning and Autonomous Robots (joint issue), 31/5, pp. 1-25, 1998.

37. S. Thrun, W. Burgard, D. Fox, and F. Dellaert, *Robust Monte Carlo Localization for Mobile Robots.* Artificial Intelligence, Vol. 128(1-2): pp. 99-141, 2001.

38. W. K. Yeap, *Towards a Computational Theory of Cognitive Maps.* Artificial Intelligence, Vol. 34: 297-360, 1988.

39. W. K. Yeap and M. E. Jefferies, *Computing a Representation of the Local Environment.* Artificial Intelligence, Vol. 107: 265-301, 1999.

Vita

Kennard R. Laviers (S'2003) was born in Hollywood, CA, in 1971. He received a BSCS degree in computer science with honors from University of Texas, El Paso, TX, in 2000. He is currently a M.S. student at The Air Force Institute of Technology, where he is researching SLAM with cognitive applications under the supervision of Professor Gilbert Peterson.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
23-03-2004	**Master's Thesis**	Oct 2002 – Mar 2004

4. TITLE AND SUBTITLE

CONCURRENT COGNITIVE MAPPING AND LOCALIZATION USING EXPECTATION MAXIMIZATION

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Laviers, Kennard R., Lieutenant, USAF

5d. PROJECT NUMBER
If funded, enter ENR #

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/04-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Mikel M. Miller AFRL/SNRP
Bldg. 620
2241 Avionics Circle
WPAFB, OH 45433-7333
(937)255-6127 ext. 4274

Capt Chris Brann AFRL/MNGN
101 West Eglin Blvd
Eglin AFB, FL 32542
(850)882-5388 ext. 1291
DSN: 872-5388 ext. 1291

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Robot mapping remains one of the most challenging problems in robot programming. Most successful methods use some form of occupancy grid for representing a mapped region. An occupancy grid is a two dimensional array in which the array cells represents *(x,y)* coordinates of a cartesian map. This approach becomes problematic in mapping large environments as the map quickly becomes too large for processing and storage.

Rather than storing the map as an occupancy grid, our robot (equipped with ultrasonic sonars) views the world as a series of connected spaces. These spaces are initially mapped as an occupancy grid in a room-by-room fashion using a modified version of the Histogram In Motion Mapping (HIMM) algorithm extended in this thesis. As the robot leaves a space, denoted by passing through a doorway, it converts the grid to a polygonal representation using a novel edge detection technique. Then, it stores the polygonal representation as rooms and hallways in a set of Absolute Space Representations (ASRs) representing the space connections. Using this representation makes navigation and localization easier for the robot to process. The system also performs localization on the simplified cognitive version of the map using an iterative method of estimating the maximum likelihood of the robot's correct position. This is accomplished using the Expectation Maximization algorithm. Treating vector directions from the polygonal map as a Gaussian distribution, the Expectation Maximization algorithm is applied, for the first time, to find the most probable correct pose while using a cognitive mapping approach.

15. SUBJECT TERMS

Robot mapping, Occupancy grid, Histogram In Motion Mapping, Absolute Space Representations, Expectation Maximization, Cognitive mapping, Localization, Polygonal map

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Gilbert L. Peterson, Dr., USAF (ENG)
U	**U**	**U**	**UU**	98	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext. 4281, e-mail: gilbert.peterson@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18