MONTE CARLO LOCALIZATION FOR ROBOTS USING DYNAMICALLY EXPANDING OCCUPANCY GRIDS

by

KARAN M. GUPTA, B.E.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

Larry D. Pyeatt
Chairperson of the Committee

Richard Watson

Accepted

John Borrelli
Dean of the Graduate School

May, 2005

ACKNOWLEDGMENTS

The course of my graduate education has been a long and winding one. All through this study I have seeked for and always found good advice from my Advisor, Dr. Larry Pyeatt. I would like to thank him for the time he took out to listen to some of the ideas that I had and the valuable feedback that he gave. I am also deeply thankful to Dr. Richard Watson for so kindly having consented to being on my Thesis Committee.

I am grateful to my good friends and colleagues Ajay Bansal and Krishnan Pazhayanoor for those critical inputs that have helped me during the course of this research. Special thanks go to Jackie Moore, for proof-reading this document and for those jovial moments in the lab.

I would like to thank my close friends Sunny, Ranjit, Ashish, Amit, Sumit and Miki for standing by me at all times, and for always believing that I can achieve much more than what I believe I am capable of. I am grateful for their encouragement and for those several more lighter moments.

My sincere thanks to Rashmi, for being a constant source of motivation and for helping me maintain focus on the task at hand, whenever I would digress.

During the course of my education, from learning the basics to Graduate School, my Family has always been first and foremost in supporting me. I would like to take this opportunity to whole-heartedly thank them for bringing me up the way they have, and for providing me the best education that I could get. It is their continued faith in me, because of which I now feel ready to implement all that I have learnt from them to the greatest exam we all face: Life.

TABLE OF CONTENTS

ABSTRACT

The past few years have seen tremendous growth in the research areas of Mobile Robotics. While growth has been fast and several problems have been very splendidly solved most mobile roboticists are faced with two primary challenges: how will the robot gather information about its environment and how will it know where it is? These two problems are referred to as:

(i). Mapping and

(ii). Localization.

Mapping is the process whereby a robot can extract relevant information from its environment allowing it to ”remember” it. Localization is the problem of estimating a robot’s pose relative to a map of its environment. However, both these problems are computationally intensive to solve and furthermore, limitations on a robot’s on board computational abilities and inaccuracies in sensor hardware and motor effectors make it even harder. Most mapping techniques are limited by memory and hence a robot has a limitation on the area that it can directly map. Also, if the mapped area is extended, most mapping implementations require that the mapping parameters be changed and the entire mapping algorithm be executed again. However, in recent times a new mapping technique was explored which is that of using Dynamically Expanding Occupancy Grids (Ellore 2002), and of using a Centralized Storage System (Barnes, Quasny, Garcia, and Pyeatt 2004). By using this approach, the robot has virtually unlimited storage space, limited only by the hard drive space, and a small initial map which grows as the robots explores its environment.

Localization has not yet been attempted using Dynamically Expanding Occupancy

Grids and a Centralised Storage System. This research is geared towards implementing Monte-Carlo Localization methods (Fox, Burgard, Dellaert, and Thrun 1999; Dellaert, Fox, Burgard, and Thrun ; Thrun, Fox, Burgard, and Dellaert 2001; Fox, Thrun, Burgard, and Dellaert 2001) to robots using Dynamically Expanding Occupancy Grids. By using this approach this research aims to provide a complete mapping and localization implementation for robots using dynamically expanding occupancy grids and a centralized storage system.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

There are three key problems that a mobile robot has to deal with:

1. Path-Planning

   Given a particular goal, a robot must be able to generate a path that it will follow from its current position in the environment to the specified goal.

2. Map Making

   A mobile robot must be able to "map" its environment. A mobile robot can perceive its environment by using various sensors, such as lasers, sonar, infrared, etc. to gather data about its environment which the robot can understand and store in a suitable data structure. This allows the robot to "remember" its environment, thereby creating a map of the environment. This map is used by the path-planner to decide a suitable route for the robot to reach its destination on the map from the robot's current position.

3. Localization

   To be able to navigate reliably the robot needs to know where it is, with respect to its environment; this is referred to as localization. Whether it is to build a map or to generate a path, the robot must know where it is.

A mobile robot perceives its surroundings with the help of sensors, varied, not only in scope and resolution, but also in their accuracy, performance and reliability. Increasing performance in terms of hardware is difficult since there are physical limits and inherent points of failure to any system. These pitfalls are often compensated by adept application of solid theory and ingenious techniques. One such method, with respect to mapping in mobile robotics, is the use of probabilistic techniques to store and validate the sensor readings(Elfes 1989a; Elfes 1989b; Martin and Moravec 1996).

Mobile robots often receive perceptions as numerical proximity values. It is the task of the *cartographer* module within the robot to convert such numerical values into true representations of the world. Occupancy grid mapping, proposed by Moravec and Elfes (Moravec and Elfes 1985; Moravec 1988), is such a technique which provides an efficient internal representation of static environments using ultrasonic range measurements. Dynamically Expanding Occupancy Grids, first proposed by Ellore (Ellore 2002), combined with Centralized Storage Systems, described by Barnes et al. (Barnes, Quasny, Garcia, and Pyeatt 2004) is an efficient method for removing the data storage limitations on a robot and allowing it to map an environment of any size.

The goal of this research is to implement Monte Carlo Localization on a robot using Dynamically Expanding Occupancy Grids for representing the data, as well as using a Centralized Storage System for storing and sharing this information between the different modules of the system.

Chapter 2 describes in detail the occupancy grid method of maintaining sensor information and creating a two dimensional map. A subsection of this chapter also explains dynamically expanding occupancy grids and how they supplement the already existing occupancy grid approach. Chapter 3 explains the sonar model and the application of Bayes

Theorem to the sonar model. Chapter 4 describes the development environment and simulator used in this research. The mapping algorithm used in this research is detailed in Chapter 5. Chapter 6 describes Monte Carlo Localization and how it is used to localize a robot. The research methodology and implementation details are discussed in Chapter 7. Finally, the conclusion in Chapter 8 details the limitations of this implementation and discusses future work that this research could lead to.

CHAPTER 2

OCCUPANCY GRIDS

Occupancy grids, also known as Evidence grids or Certainty grids were first proposed by Moravec and Elfes (Moravec and Elfes 1985) and formulated in CMU (Martin and Moravec 1996). Occupancy grids provide a formal method for fusion of sensor data, similar to those with ultrasonic range sensors (Elfes 1989a; Elfes 1989b; Moravec and Blackwell 1992). Evidence grids provide a formal sensor independent representation schema for mapping. Occupancy Grids have been implemented with laser range finders, stereo vision sensors (Moravec 1996) and even with a combination of SONAR, infrared sensors and sensory data obtained from stereo vision (Lanthier, D.Nussbaum, and A.Sheng 2004).

2.1 Regular Occupancy Grids

As introduced above, Occupancy Grids provide a data structure that allows for fusion of sensor data. It provides a representation of the world which is created with inputs from the sensors. They can be viewed as a two dimensional array of probability values, as shown in Figure 2.1, with each grid representing a user-defined discrete location in the robot's world. These probabilities are based upon the hypothesis that a given grid is either occupied or empty:

$$P(Occupied) = 1 - P(Empty). \tag{2.1}$$

Since the probabilities are identified based on the sensor data, they are purely conditional. Hence, the occupancy grid holds probabilities of a grid being occupied or empty *given* a sensor data. Thus, the above equation can be rewritten as

$$P(s|Occupied) = 1 - P(s|Empty) \tag{2.2}$$

$$P(s|Occupied) + P(s|Empty) = 1. \tag{2.3}$$

If the sensor reading is a range reading obtained from a sensor, such as SONAR or LASER, the term *grid* means imaginary checkered zones on the terrain the robot traverses. Occupancy grids are used with this context throughout the scope of this research.

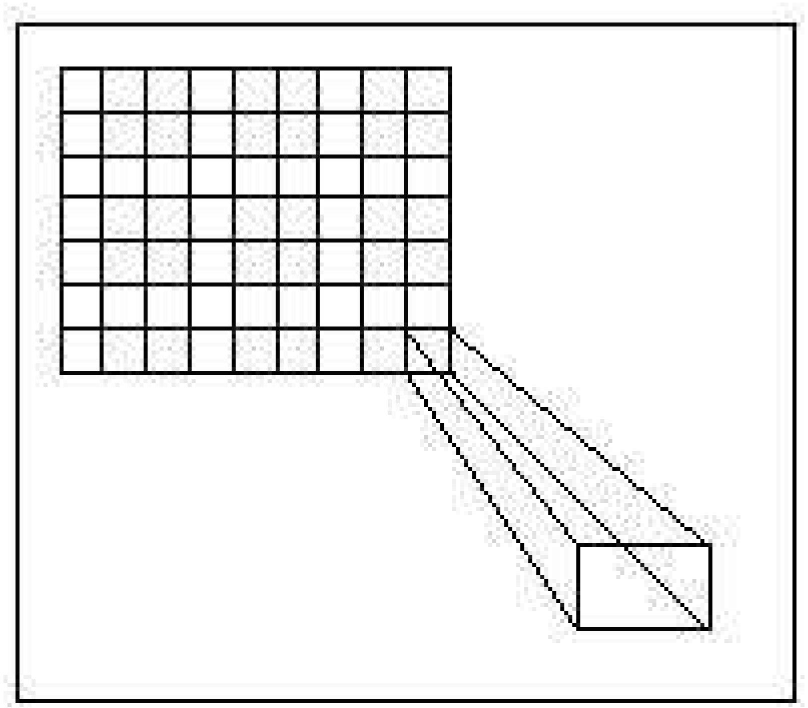


Figure 2.1: A Regular Occupancy Grid

Apart from being used directly for sensor fusion, there also exist interesting variations of Evidence grids, such as place-centric grids (Youngblood, Holder, and C 2000), histogram grids (Koren and Borenstein 1991) and response grids (Howard and Kitchen 1996). The variations will not be studied in this research.

2.2 Dynamically Expanding Occupancy Grids

Explored by Ellore (Ellore 2002) in his Master's thesis, dynamically expanding occupancy grids provide a much needed respite from having to restrict robot maps to the memory provided by the robot.

With conventional occupancy grid methods, the occupancy grid map is stored in memory and updated as the robot moves around its environment. By using dynamically expanding occupancy grids, we free the robot from dependency on the memory that it is provided. With this technique, only a small section of the map needs to be stored in memory, whereas the rest can be stored on disk. A Centralised Storage System provides a convenient storage mechanism to save the occupancy grid, and share it with other modules of the robot, such as the Localizer and the Path-Planner.

When using Dynamically Expanding Occupancy Grids (DEOGs) to map its environment, the robot initially assumes a uniform distribution of probability over the area it will traverse. This assumption signifies an unexplored, unknown environment. As the robot moves, it gets perceptual information from its sensors, and odometric information from its motors. The cartographer processes this data and adds grids to the map (which was initially a single grid: the starting location), as it discovers previously unexplored locations. At every update phase, the cartographer may either update the belief in an already existing

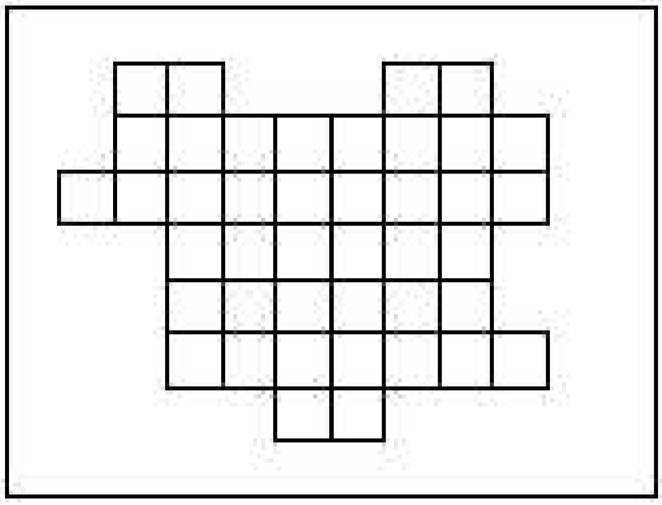


Figure 2.2: A Dynamically Expanding Occupancy Grid

grid, or it may add a new grid to the map with the newly computed belief. The computation of the occupancy belief for a particular grid on the map is done using Bayesian Estimation which is discussed in Chapter 3. One can see how a DEOG differs from a regular occupancy grid by looking at Figure 2.2. Once the robot has gained sufficient belief (either an occupancy belief or a belief in emptiness) for each grid cell, over the entire environment, mapping is complete. This map can now be used by the Localizer to successfully localize the robot in its environment. As the map is stored in a central location, the localizer can now use this data in any way that is suitable to the programmer. The entire map may be downloaded into the robot's memory (for very fast access), or it may be downloaded in sections as the robot wanders in the environment. The localizer may also update the map as it gathers new data from the world, thereby improving the map as well.

How this research implements Occupancy Grids as Dynamically Expanding Occupancy Grids using a Centralised Storage System, for mapping and localization, is described in detail in Chapter 7.

2.3 Merits and Demerits

With a strong convergence and absence of local minima, as with the use of Potential fields (Arkin 1989; Koren and Borenstein 1991), Occupancy Grids is often a preferred technique used in robotics. The computation is incremental and accommodates for any sonar noise unlike techniques such as Kalman and Multi-planar maps which assume a Gaussian noise (Thrun 2002). With conventional occupancy grids, the dimensionality of the map is limited only by the robot's hardware. As demonstrated in this research and others there are different techniques used in conjunction with conventional occupancy grids to provide it with better data handling. Use of dynamically expanding occupancy grids, first explored by Bharani (Ellore 2002) and multi-agent mapping with a database back-end (Barnes, Quasny, Garcia, and Pyeatt 2004) are good examples of extensibility of Occupancy Grids.

In spite of its ability to handle uncertainty by using probabilistic methods, Occupancy Grids have a few shortcomings, such as a lack of method for accommodating pose uncertainty. A second inadequacy is the assumption of independent noise. The assumption not only prevails with sonar data but also extends to the assumption of independence of adjacent cells. This problem has often been overcome by discarding data when the robot is stationary.

To summarize, albeit with minor deficiencies, Occupancy grids provide a robust probabilistic vehicle for sensor fusion and real world representation with very simplistic implementation requirements.

2.4 Applications

Occupancy Grids present a human readable and lucid representation of real world objects. They have been extensively used for mapping (Moravec 1996; Howard and Kitchen 1997; Burgard, Fox, Jans, Matenar, and Thrun 1999; Lanthier, D.Nussbaum, and A.Sheng 2004; Cahut, Valavanis, and Delias 1998; Wallner and Dillmann 1994; Chong and Kleeman 1999; Konolige 1997; Dedieu and del R. Millain 1998). Occupancy grids have also been successfully used for localization (Yamauchi, Schultz, and Adams ; Balch 1996; Yamauchi and Beer 1996; Olson 1999; Thrun, Fox, Burgard, and Dellaert 2001; Dellaert, Fox, Burgard, and Thrun) and position tracking (Burgard, Fox, and Henning 1997; Schiele and Crowley 1994). The evolution of Occupancy Grids as a popular technique for the fundamental processes of robotics could be attributed to its simplicity, probabilistic iterative approach and robustness.

Chapter 3 describes in detail how sensor information is converted into probabilistic occupancy values in a discrete occupancy grid.

CHAPTER 3

THE SONAR MODEL

A sonar is a system which sends out sound pulses and calculates distance to an object based on the time it takes for the object to reflect back the sonar beam to the transducer. A sonar model is used to chart this distance data, gathered by a sonar into a representation of an object being present, or not being present, in the path of the sonar beam. The sonar model provides us with a range reading about the area around the robot, and a suitable program can use this information to mark the area as having a certain probability of being EMPTY or OCCUPIED.

The robot simulator used in this research is for the Nomadic Technologies' Super Scout II. It is equipped with a ring of 16 Polaroid 6500 sonar ranging modules. Each sonar module can detect an object within a range of 6 inches to 35 feet, with an accuracy of $\pm1\%$.

3.1 Sonar Regions

The robot observes its surroundings with the help of its sensors. The sensor component is an array of 16 SONARS as seen in Figure 3.1, with the orientation following the arrow. Each sonar returns a range reading which gives the distance to the nearest obstacle or the range of the sonar itself, if there is no obstacle in the field of view of the sonar. The readings are integer values and are specific to a given orientation and position. A typical visualization of these range readings is depicted in Figure 3.2

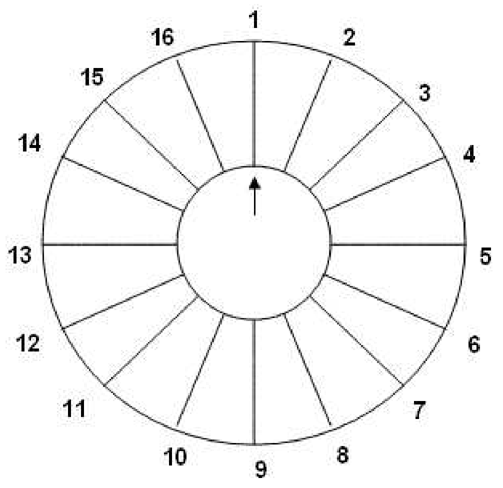


Figure 3.1: Arrangement of SONAR array on Simon

Such measurements cannot be directly used in cartography or localization. A sonar model needs to be constructed to interpret the range readings. Sonar models are derived in numerous ways. *Empirical methods* depend on beliefs in observations and model constructed from such beliefs. Beliefs are accumulated by collecting data and verifying with real world data. *Subjective methods* are similar to empirical methods but rely on assumed beliefs instead of collecting data and using it to construct the model. In the case of Occupancy Grids, the one dimensional range reading needs to be converted to a two dimensional array of evidence grids pertaining to the flat surface of the ground. Here, the physical properties of the sensor are used to generate the model. This method thus comes under the third category called *Analytical Methods*. Sonar models have been implemented empirically using methods like neural nets (Ellore 2002; Thrun 1998). This research will follow the standard analytical sonar model as described in (Murphy 2000).

A typical implementation of this sonar model is as shown in Figure 3.3. The figure shows the model of a single beam of sonar data. The field of view of the beam is given by

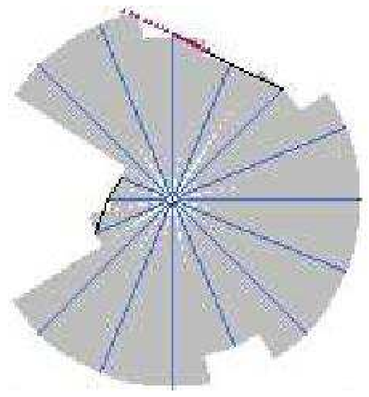


Figure 3.2: Typical range readings obtained from SONAR at one instance

the value β, and maximum range the sonar can measure is given by R. For every point on the beam projected by the sonar, the angle it makes with respect to the sonar is measured as α and it distance from the sonar is given by r. Typically, a collection of such adjacent points forms the occupancy grids (discussed in Chapter 2). The points on this beam can be classified as below:

- Region I: is the locus of all points equidistant from the sensor with distance equal to the range reading returned. This region signifies the area where the points are probably occupied, based on the current sonar reading.

- Region II: is the collection of all points which are probably empty. This region is found between the sonar and Region I within the beam.

- Region III: consists of points which have no information regarding their state, or in other words, are neither occupied nor empty.

Given the beam of the sonar as in Figure 3.3, it can be divided into adjacent elements of an occupancy grid. It is evident that, when a range reading is received, the obstacle

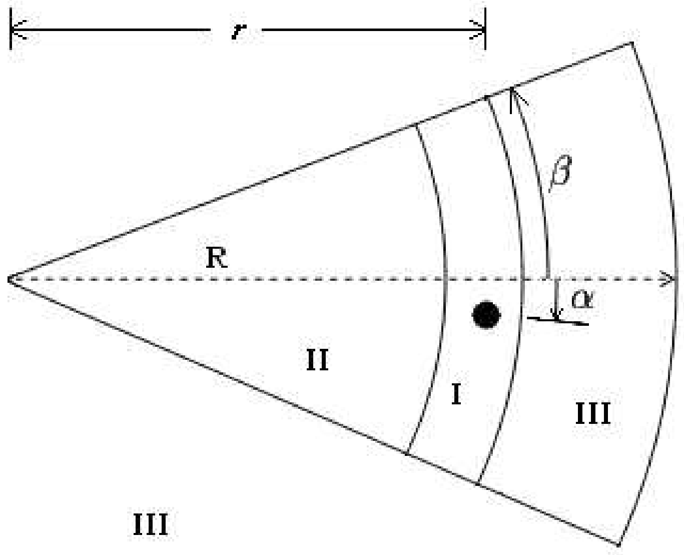


Figure 3.3: The classification of Regions in the Sonar Model

indicated by the range reading could be anywhere in the area denoted by Region I. Yet, the probability of finding the obstacle is greater via a direct reflection of the sonar for that distance. Hence, $P(Occupied)$ is highest at the axis of the sonar (α=0) and the probability reduces as $\alpha \to \beta$. Another heuristic applied is that, given a range reading, the error is much less when the detected obstacle is nearer to the sonar than when the obstacle is farther away. Hence, the updates are *stronger* for grids closest to the sonar and tend towards uncertainty

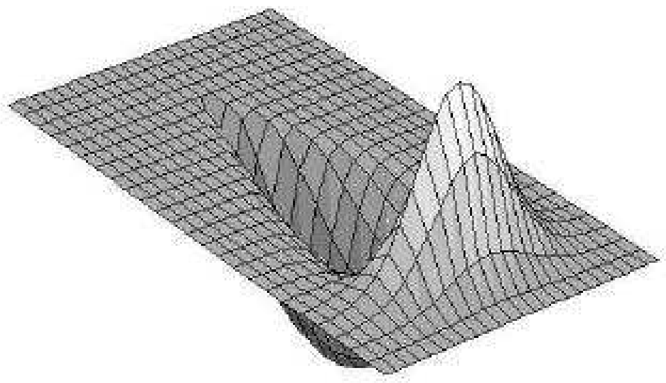


Figure 3.4: Visualization of the probabilities effected by a single sonar reading

as $r \to R$. Thus, for every element in **Region I**:

$$P(Occupied) = \frac{\left(\frac{R-r}{R}\right) + \left(\frac{\beta-\alpha}{\beta}\right)}{2} \times Max_{occupied} \tag{3.1}$$

$$P(Empty) = 1.0 - P(Occupied). \tag{3.2}$$

The update equation for elements in **Region II** is given by:

$$P(Occupied) = 1.0 - P(Empty) \tag{3.3}$$

$$P(Empty) = \frac{\left(\frac{R-r}{R}\right) + \left(\frac{\beta-\alpha}{\beta}\right)}{2}. \tag{3.4}$$

The factor of $\frac{R-r}{R}$ in the above equations increases belief in elements nearer to the sonar, and the factor $\frac{\beta-\alpha}{\beta}$ increases belief along the tangent to the surface of the robot at the point where the sonar is present. The $Max_{occupied}$ is the maximum belief assigned to the sonar reading. A value slightly below 1.0 is typically used for $Max_{occupied}$.

Given that the beam is overlaid on an Occupancy Grid, the sonar model looks similar to Figure 3.4. The elevation indicates a high probability of the grid element being occupied and the depression signifies a high probability of being empty.

3.2 Bayesian Updates

The equations from section 3.1 provide us with the probability of the hypothesis H where H = {Occupied,Empty} or H = {H,¬H} and $0 \leq P(H) \leq 1$ Again, by the basic property of probability, $P(\neg H) = 1 - P(H)$. The probabilities $P(H)$ and $P(\neg H)$ are *unconditional probabilities*. Unconditional probabilities only provide *a priori* information and hence sensor data cannot be fused using them. Bayes' rule provides a mathematical solution for this problem. Bayes rule can be derived as below:

Let $P(A)$ be the probability that the event A occurs and $P(B)$ be the probability of occurance of event B. $P(A \cap B)$ is the probability of occurance of A and B and $P(A|B)$ is the probability of occurance of A given that the event B has occurred. Thus, by the product rule:

$$P(A \cap B) = P(A) \times P(B|A). \tag{3.5}$$

Similarly, Equation 3.5 can be written as:

$$P(B \cap A) = P(B) \times P(A|B). \tag{3.6}$$

But we know that,

$$P(A \cap B) = P(B \cap A). \tag{3.7}$$

Using in equations 3.5 and 3.6 and solving for $P(A|B)$,

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}. \tag{3.8}$$

This can be generalized as

$$posterior = \frac{conditional\ likelihood \times prior}{total\ likelihood}. \tag{3.9}$$

Where, the prior is $P(H|s)$, calculated as formulated by the equations in Section 3.1, conditional likelihood is the likelihood of observations given the element is said to be occupied i.e., $P(s|H)$. and the likelihood in the denominator is a normalizer. It is calculated as a summation of all possible H and with respect to the sonar model, is given by $P(s|H)P(H)+P(s|\neg H)P(\neg H)$ The posterior obtained from Equation 3.9 hence gives the conditional probability that a given element in the occupancy grid is occupied.

Ideally, the above equation would be sufficient if every element were to be updated only once throughout the life cycle of mapping. The likelihood of such an occurance is extremely rare. Occupancy grids, with Bayesian updates is an incremental method. Hence, a single point could be updated with multiple iterations of s. Accommodating this, Equation 3.9 becomes

$$P(H|s_1,s_2,...s_n) = \frac{P(s_1,s_2,...s_n|H)P(H)}{P(s_1,s_2,...s_n|H)P(H)+P(s_1,s_2,...s_n|\neg H)P(\neg H)}. \tag{3.10}$$

Assuming independence of sonar readings,

$$P(s_1,s_2,...s_n|H) = P(s_1|H)P(s_2|H)...P(s_n|H). \tag{3.11}$$

Now, memory of $P(s_n|H)$ for n observations is computationally expensive. Fortunately, the rule

$$P(A|B)\times P(B) = P(B|A)\times P(A). \tag{3.12}$$

can be applied to Equation 3.10 and a recursive version of the rule can be derived:

$$P(H|s_n) = \frac{P(s_n|H)\,P(H|s_{n-1})}{P(s_n|H)\,P(H|s_{n-1}) + P(s_n|\neg H)\,P(\neg H|s_{n-1})}. \tag{3.13}$$

Thus, with every sonar reading Equation 3.13 can be used to update existing probabilities with new conditional probabilities. Besides the Bayesian update mechanism, there also exist other methods for converting sonar data into a belief value for an occupancy grid. These include the Dempster-Shafer method (Murphy 2000) which is based on work by Shafer (Shafer 1976), and Histogrammic In-Motion Mapping (HIMM) which was developed by Borenstein and Koren (J. Borenstein 1991; Murphy 2000).

CHAPTER 4

EXPERIMENTAL SETUP

4.1 Experimental Robot

The research was performed on the simulator for the Super Scout, shown in Figure 4.1, an integrated mobile robot system with ultrasonic, tactile and odometry sensing. There are two wheels on the sides and one caster wheel for support. The side wheels are used as effectors and are responsible for the motion of the robot. The wheels are used for both linear motion and reorientation. The robot is equipped with the Nomadic Technologies Sensus200 sensor device which forms a ring of 16 Polaroid 6500 sonar ranging modules. Each sonar module has the ability to return range readings from 6 inches to 35 feet, with an accuracy of ±1% over the entire range (Nomadic Technologies Inc. 1999). The firing order and rate of the modules is configurable via the API.

The simulator is the Nomadic Host Software Development Environment (Nomadic Technologies Inc. 1999b). It is a client-server based system that controls and simulates robotic environments for the family of Scout robots provided by Nomadic Technologies *(Copyright 1992-1998 Nomadic Technologies, Inc)*. The client application can either run the robot through a *simulated* environment or can pass on the commands to the real robot. The Server is a convenient way to send commands to the robot and to receive sensing data from the robot. It provides an elaborate graphic interface and simulation capabilities. The server is run as a separate process on a workstation. The server process communicates

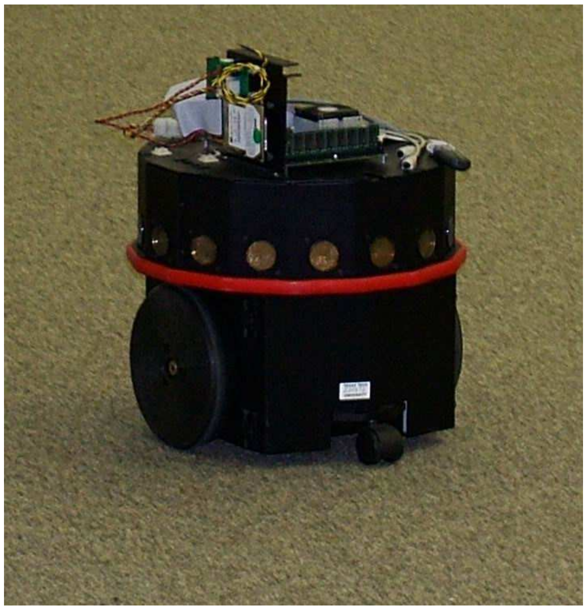


Figure 4.1: Simon - the Nomadic Super Scout II

with the robot process by TCP/IP. The server is configured to be in simulation mode by default, so that actions do not have any physical consequences, until the real-robot mode is manually selected by the user. The simulator provides support for true multi-agent system research, allowing multiple robots to be concurrently active in the same world. A daemon is continuously running on the robot, which accepts commands from the outside (the server, a joystick, etc.), executes them, and send back any results.

4.2 The Graphic Interface

The Graphic User Interface (GUI) provides a convenient access to the real and simulated robots, and to the representation of the world, as shown in Figure 4.2. Through the GUI, the user can send commands to the robot, monitor command execution by looking at the robot movements on the screen, and also view instantaneous and cumulated sensor data. The user can also create and modify a simulated environment, and use it to test robot programs.

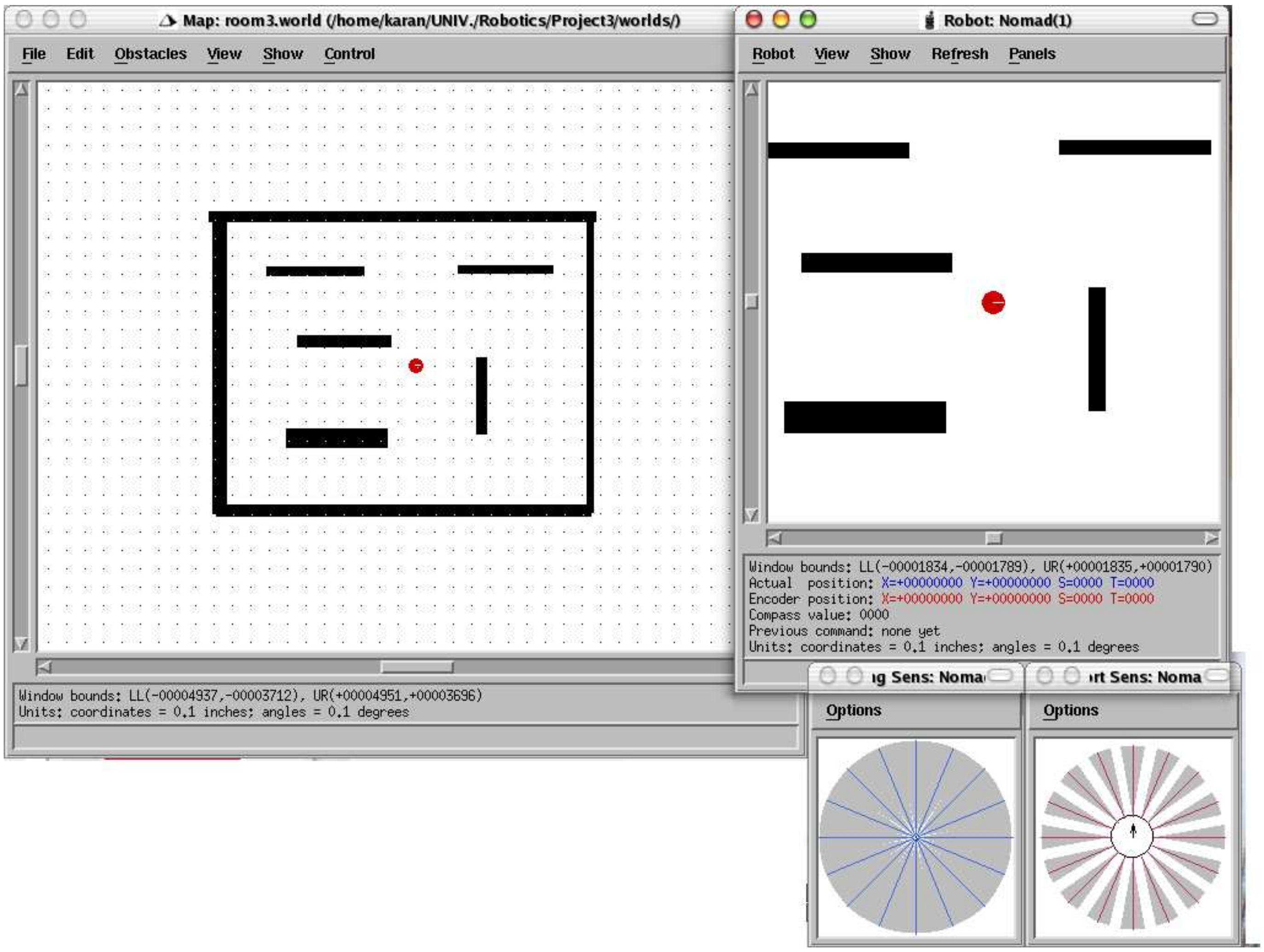
The GUI has the following 4 main windows:

- The Map Window
- The Robot Window
- The Short Sensors Window
- The Long Sensors Window

There is only one Map window, but as many Robot windows as there are robots (real or simulated). The Map window gives access to the world representation, with functions such as: creating/deleting obstacles, editing obstacles, etc. The Robot window allows the display of sensor history, the path of the robot, as well as the execution of robot commands. Both windows support usual display functionalities like zooming, scrolling, centering, etc. Several display parameters for the Map window are controlled by values set up in the world.setup file, which is read by the server when it starts up. Display parameters for the Robot window are stored in the robot.setup file.

4.2.1 The Simulator

The graphic display is the same for both a simulated or a real robot. By default the GUI is in simulation mode. The Robot item in the Robot window menu has two options:



```
Window bounds: LL(-00004937,-00003712), UR(+00004951,+00003696)
Units: coordinates = 0.1 inches; angles = 0.1 degrees
```

```
Window bounds: LL(-00001834,-00001789), UR(+00001835,+00001790)
Actual  position: X=+00000000 Y=+00000000 S=0000 T=0000
Encoder position: X=+00000000 Y=+00000000 S=0000 T=0000
Compass value: 0000
Previous command: none yet
Units: coordinates = 0.1 inches; angles = 0.1 degrees
```

Figure 4.2: The Four Main Windows of The Robot Simulator

Real Robot or Simulated Robot, one of which is selected to toggle between the real and simulated modes.

4.2.2 The Map Window

The Map window allows an application to interactively define and modify the map of the world where a robot moves around. The robot world is an abstract coordinate system; its dimensions are set in the world.setup file under the [physical ¿ size] entry. The two pairs of coordinates at the bottom of the window reflect the current positions of the window's lower-left corner and the upper-right corner in the world, respectively. Initially the (0, 0) point of the coordinate system is the center of the Map window. This window has six menus: File, Edit, Obstacle, View, Show and Control. The File menu is used to load/save maps from/to files. The Edit menu allows editing of obstacles on the map. The Obstacles menu allows adding/removing/moving of obstacles on the map. The View menu has the basic view commands to zoom, center, etc. The Show menu enables us to display and hide various items in the window: the Map and the robots. The Control menu has commands to create a robot, and to control the speed of simulation.

4.2.3 The Robot Window

There are as many Robot windows as there are robots in the Map window. The robot window allows interactive control of a robot. This window has 5 menus: Robot, View, Show, Refresh and Panels. The bottom of the window displays the status of the robot: the current position of the robot, the compass value, and the last command issued. Position Information is displayed as X, Y, S and T, where X and Y are the coordinates, S is the steering direction and T is the Turret direction. On the robot used in this research, the Super Scout, the turret always faces the same direction as the steering angle. Degrees range from 0 to 360, with 0 as the horizontal right. The Robot menu has the option of switching between the real and simulated robot modes. It also allows placement of the robot anywhere in the world. The View menu is about the same as the one in the Map window. The Show menu displays/hides the trace of the robot as it moves, and the various sensors. The Refresh menu allows for clearing the data collected and being displayed on the window. The Panels menu allows for direct control of the robot, by providing a graphical joystick and a command panel, as well as a recorder.

4.2.4 The Short Sensors Window

This window displays the data being perceived by the short-range sensors on the robot, i.e. the Infrared Sensors and the Bumper Sensors. It displays the bumpers when the robot runs into an obstacle and displays the infrared sensor data as conical radius lines. Since the Scout does not have Infrared sensors, the IR radius lines are meaningless here. There are two kinds of viewing modes in this window: Local View and Global View. In the local view, the robot's forward direction is always aligned with the upward vertical direction of the window. In the global view, the robot will rotate in the window, as the robot rotates in the world, with respect to the environment.

4.2.5 The Long Sensors Window

This window displays the data from the Sonars as radius lines. It has an Options menu which allows us to modify the Sonar Sensor display, from cones and conical arcs, to only straight radius lines, to no display at all. It also displays the data from a Laser sensor, but since the Scout does not have a Laser, nothing is displayed for the Laser. Similar to the Short Sensors Window, this window also has options for displaying the robot in a Local View or a Global View.

4.3 Programming The Robot

Control commands may be sent to the robot either manually, or via a program, which connects to the robot's server.

There are two ways to control the robot from a program:

- Direct Mode The program communicates directly with the robot daemon.
- Client Mode The program behaves as a client and connects to the server. The server accepts the commands from the program, just like it does from the GUI, and transmits them to the robot daemon. The server can also transmit commands to a simulation module of the real robot.

The client mode is used for testing and debugging a program. The direct mode is preferable in the production stage, when the program to be run on the robot has been thoroughly tested, to minimize communication overhead.

4.4 The Global State Vector

The robot maintains a global array called the State Vector. Information about the current state of the robot, its position, readings from the sensors, etc. are accessible to an application program when it reads this State Vector. This vector is updated upon the execution of every robot command which causes a change in the robot's configuration. The `gs()` (getstate) can be issued to forcefully update the State Vector. The fields defined in the State Vector are listed in Table **??**.

CHAPTER 5

MAPPING

In the context of mobile robotics, Mapping is the process whereby a mobile robot must translate the environment, using some mathematical model, into a form which it can later refer to. The map thus stored, can be used for localization and path-planning.

Localization and mapping are very closely related: a robot will not be able to map very well if it does not know where it is. And if the robot does not have a map, then it will be very difficult for the robot to localize itself. Hence, localization cannot be accomplished without a good mapping algorithm.

There are several mapping techniques that are in use by roboticists at this time. Some of them involve *feature extraction* from the environment and using that knowledge to create a map. This process involves detecting *landmarks* and *gateways*. A landmark is a perceptually distinctive feature of interest on an object or in a locale of interest (Murphy 2000). When the robot wanders in its world, and it finds a landmark, such that the landmark is marked on the map as well, then the robot can safely localize itself with respect to the map. Another approach is using *Relational methods* (Murphy 2000) where the world is represented as a graph or network of nodes and edges. Nodes represent gateways, landmarks or goals and the edges represent a navigable path between two nodes. Extra information may be associated with the edges and nodes based on the implementation.

This research is using a Configuration Space representation of the world, i.e. an Occupancy Grid. In this approach, the robot uses a data structure which holds the position (location and orientation) of any objects and itself, in the world. Besides Occupancy Grids, other Cspace approaches include Meadow maps, Voronoi diagrams, Quadtrees and Octrees. Please see Murphy (Murphy 2000) for a detailed explanation of these approaches.

As described in Chapter 2, the occupancy grid approach involves hypothetically superimposing a two-dimensional Cartesian grid on the world space. Any grids which are over the are occupied by an object are marked as *occupied* and those which are not over any object are marked as *empty*. The Bayesian updation mechanism discussed in 3 is used to mark the belief value of each grid.

CHAPTER 6

MONTE CARLO LOCALIZATION

6.1 Localization

Mobile robot localization is the problem of estimating a robot's pose (location, orientation) relative to its environment. It is a key problem in mobile robotics since, it plays a pivotal role in various successful mobile robot systems and has been referred to as "the most fundamental problem to providing a mobile robot with autonomous capabilities" (Thrun, Fox, Burgard, and Dellaert 2001). Most successful mobile robot systems to date utilize some type of localization, as knowledge of the robot's position is essential for a broad range of mobile robot tasks.

There are three different aspects to mobile robot localization:

1. *Position Tracking*

   In position tracking, the initial pose of the robot is known and the problem is to compensate small, incremental errors in a robot's odometry.

2. *Global Positioning*

   In the global localization problem, the robot does not know its initial pose, and it has to determine its correct location from scratch.

3. *Kidnapped Robot Problem*

   This is the most difficult localization problem. In this situation a robot is lifted from its current known location and moved somewhere else, without being told. Now the robot has to relocalize itself to the new location, while it firmly believes that it is somewhere else. This problem is generally used to test a robot's ability to recover from catastrophic localization failures.

All of these problems become even harder to solve in dynamic environments, such as in the proximity of people who affect the robot's sensor measurements in unpredictable ways.

The vast majority of existing algorithms address only the problem of position tracking. The nature of small incremental errors makes algorithms such as Kalman filters applicable. Kalman filters have been successfully applied in a range of fielded systems. Kalman filters estimate posterior distributions of robot poses conditioned on sensor data. Exploiting a range of restrictive assumptions, such as Gaussian-distributed noise and Gaussian-distributed initial uncertainty, they represent posteriors by Gaussians, exploiting an elegant and highly efficient algorithm for incorporating new sensor data. However, the restrictive nature of the belief representations makes them inapplicable to global localization problems in which single Gaussian does not accurately reflect the true distribution.

This limitation is overcome by two related families of algorithms: localization with multi-hypothesis Kalman filters and Markov localization. Multi-hypothesis Kalman filters represent beliefs using mixtures of Gaussians, thereby enabling them to pursue multiple, distinct hypotheses, each of which is represented by a separate Gaussian(Kalman 1960). However, this approach inherits the Gaussian noise assumption from Kalman filters. To meet this assumption, virtually all practical implementations extract low dimensional features from the sensor data, thereby ignoring much of the information acquired by a robot's sensors

Markov localization algorithms, in contrast, represent beliefs by piecewise constant functions (histograms) over the space of all possible poses. Just like Gaussian mixtures, piecewise constant functions are capable of complex, multi-modal representations. However, accommodating raw sensor data requires fine-grained representations, which impose significant computational burdens. To overcome this limitation, selective updating algorithms and tree-based representations that dynamically change their resolution have been proposed.

6.2 Monte Carlo Localization

This thesis presents a dynamic approach to performing Monte Carlo Localization (MCL). MCL solves the global localization in a highly robust and efficient way. It can accommodate arbitrary noise distributions and non-linearities. Therefore, MCL does not need to extract features from the sensor data. The MCL algorithm represents belief by a set of *samples* (also called, *particles*), drawn according to the posterior distribution over robot poses. So, rather than approximating posteriors in parametric form, as is the case for Kalman filter and Markov localization algorithms, MCL simply represents the posteriors by a collection of weighted particles which approximates the desired distribution.

As far as localization is concerned, particle representation has the following advantages over previous approaches:

1. Particle filters focus computational resources in areas that are most relevant, by sampling in proportion to the posterior likelihood.

2. Particle filters are universal density approximators, weakening the restrictive assumptions on the shape of the posterior density when compared to previous parametric approaches.

3. Particle filters are easily implemented as any-time algorithms, which are algorithms that can output an answer at any time, but where the quality of the answer depends on the time spent computing it. By controlling the number of samples, on-line, particle filters can adapt to the available computational resources. The MCL code can, thus, be executed on computers with vastly different speeds without modification of the basic code.

4. Finally, particle filters are surprisingly easy to implement, which makes them an attractive approach to mobile robot localization.

This basic MCL algorithm proposed by Thrun *et al.* (Fox, Thrun, Burgard, and Dellaert 2001; Dellaert, Fox, Burgard, and Thrun ; Fox, Burgard, Dellaert, and Thrun 1999) can be explained to consist of 4 steps:

1. For this step refer to Figure 6.1. Initially, the location of the robot is known, but the orientation is unknown. The cloud of particles Sk-1 represents our uncertainty about the robots position.

2. Robot has moved 1 meter since last time-step. We deduce that robot is now on a circle of 1m radius around the previous location. Our belief state changes to reflect this in Figure 6.2. At this point we have applied only the motion model.

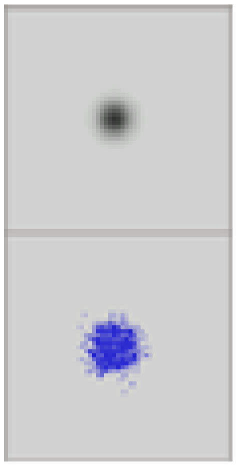


Figure 6.1: Monte Carlo Localization: Step 1

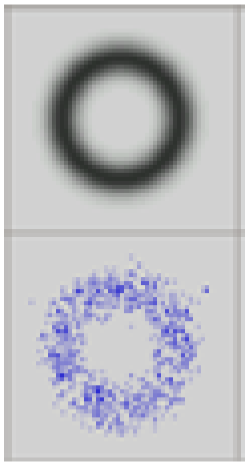


Figure 6.2: Monte Carlo Localization: Step 2

3. We now take sensor readings into account. A landmark is observed 0.5m away somewhere in the top-right corner. We apply weighting to the samples to reflect that the robot is more likely to be in the top-right corner. This weighting can be seen in Figure 6.3.

4. The weighted set is resampled to give the new set of points where the robot is most likely to be. This new set is the starting point for the next iteration. This new set of points is visible in Figure 6.4.

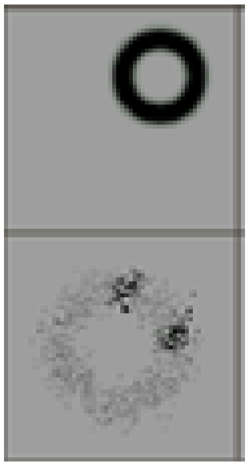


Figure 6.3: Monte Carlo Localization: Step 3

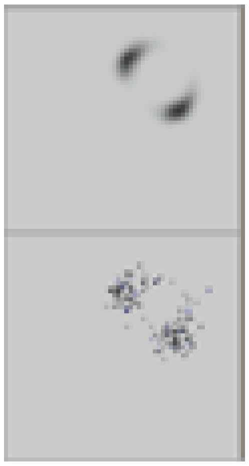


Figure 6.4: Monte Carlo Localization: Step 4

Some pitfalls too arise, due to the stochastic nature of the approximation. For example, if the sample set is small, a well-localized robot might lose track of its position just because the MCL algorithm fails to generate a sample in the right location. The basic MCl algorithm also degrades when the sensors are too accurate. Also, the basic MCl algorithm, assumes that the environment is static, and will typically fail in highly dynamic environments, such as public places where people might cover the robot's sensors for an extended period of time. These problems have been addresses in Ajay Bansal's Master's thesis (Bansal 2002) and by Thrun(Thrun, Fox, Burgard, and Dellaert 2001). This research develops upon Bansal's and extends it to a dynamic storage system.

MCL is a recursive bayes filter that estimates the posterior distribution of robot poses conditioned on sensor data. Bayes filters address the problem of estimating the state x (the robot's pose) of a dynamical system (the robot) from sensor measurements. Bayes filters assume that the environment is *Markovian*, i.e. past and future data are conditionally independent if one knows the current state. When working with mobile robots, we generally work with two kinds of data: *perceptual data* such as sonar range measurements, and *odometry data*, which is information about the robot's motion. Keeping this in mind, we can represent Bayes' posterior distribution as

$$Bel(x_t) = p(x_t | o_t, a_{t-1}, o_{t-1}, a_{t-2}, ..., o_0) \tag{6.1}$$

where x_t is the state x at time t, $Bel()$ is the *belief*, and o is the perceptual data (for observation) and a is the odometric data (for action). We assume that observations and actions arrive in an alternating sequence. We will use $a_{[t-1]}$ to refer to the odometry reading that measures the motion that occurred in the time interval $[t-1;t]$ to illustrate that the motion is the result of the control action asserted at time $t-1$.

Bayes filters estimate the belief recursively. The initial belief may characterize the initial knowledge of the system, or if such knowledge is unavailable, it is typically initialized to a uniform distribution over the state space. This uniform initial distribution represents the global localization problem, where the localizer does not know the initial pose of the robot. Bayes recursive update equation is

$$Bel(x_t) = \eta\, p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{6.2}$$

where η is a normalization constant. Together with the initial belief, it defines a recursive estimator for the state of a partially observable system. This equation is the basis for various MCL algorithms. To implement Equation 6.2 two conditional densities must be known: the probability $p(x_t|x_{t-1}, a_{t-1})$, which we will refer to as *next state density* or the *Motion Model*, and the density $p(o_t|x_t)$, which we will call *Perceptual Model* or *Sensor Model*. If the markov assumption holds, then both models are typically stationary (time-invariant), i.e. they do not depend on specific time t. So the notation can be simplified to $p(x|x', a)$ and $p(o|x)$ respectively.

For a robot operating in a plane, a pose is a three-dimensional variable $[x\ y\ \theta]$, comprising of the robot's two-dimensional Cartesian co-ordinates and its orientation. The value of a may be an odometry reading or a control command, both of which characterize a change of pose. In reality, physical robot motion is probabilistic due to many reasons like slippage in the wheels and motors, imprecise measurements and such other sources of uncertainty. Therefore, the probabilistic motion model $p(x|x', a)$, describes a posterior density over possible successors x and generalizes exact mobile robot kinematics.

For the perceptual model, $p(o|x)$, the robot depends on its sensing capabilities. Assuming that the robot's pose is x, and o_i denotes an individual sonar beam with bearing

α_i relative to the robot. Let $g(x,\alpha_i)$ denote the measurement of an ideal, noise-free sensor. Since we assume that the robot is given a map of the environment, $g(x,\alpha_i)$ can be computed using *ray tracing*. We also assume that

$$p(o_i|x) = p(o_i, g(x,\alpha_i)). \tag{6.3}$$

```
**AlgorithmMCL**$(X,a,o)$:
    $X'$ = $\phi$
    for $\iota$ = 0 to $m$ do
        generate random $x$ from $X$ according to $w_1,...,w_m$
        generate random $x' \leftarrow p(x'|a,x)$
        $w' = p(o|x',m)$
        add $<x',w'>$ to $X'$
    endfor
    normalize the importance factors $w$ in $X'$
    return $X'$
```

Figure 6.5: The MCL Algorithm

Finally, the individual density values $p(o_i|x)$ are integrated multiplicatively, assuming conditional independence between the individual measurements:

$$p(o|x) = \prod_i p(o_i|x). \tag{6.4}$$

The idea of MCL (and other particle filter algorithms) is to represent the belief $Bel(x)$ by a set of m weighted samples distributed according to $Bel(x)$:

$$Bel(x) = \{x^{(i)}, \omega^{(i)}\}_{i=1,...m}, \tag{6.5}$$

where each $x^{(i)}$ is a sample (a hypothesised state), and $\omega^{(i)}$ determine the weight *importance* of each sample. The initial set of samples represents the initial knowledge $Bel(x^0)$ about the state of the system.

The basic MCL algorithm is listed in Figure 6.5. Bansal's technique appends a filtering approach to this basic algorithm. The implementation of this algorithm with a Dynamically Expanding Occupancy Grid and a Centralized Storage System is discussed in Chapter 7.

CHAPTER 7

RESEARCH METHODOLOGY

7.1 The Centralized Storage System

To implement Dynamically Expanding Occupancy Grids, the robot system needs a central location to store discovered grids. It must also be able to quickly and smoothly add grids to the occupancy grid. This central location and its access methods make up the Central Storage System (CSS).

In this implementation, the CSS is an *Oracle 9i* database. The robot system's modules connect to and access the database by linking the OTL library based on templates. This research uses version 4.0 of the library. The robot modules (the cartographer, localizer) designed in this research use a common database header file. This file, `db.h`, is the heart of the CSS and is described in detail in Section 7.1.3.

7.1.1 The OTL Library

The *OTL (Oracle Template Library)* is a C++ library developed by Sergei Kuchin (Kuchin). It consists of a single header file `otlv.h` which needs to be included in the application programs. The OTL code gets expanded into direct database API function calls, so it provides ultimate performance, reliability and thread safety in multi-processor environments as well as traditional batch programs. OTL 4.0, being a template library, is highly portable since it is self-sufficient and compact enough (around 380Kb in a single header file). The OTL library is available for download from `http://otl.sourceforge.net/`. OTL 4.0 is ANSI C++ compliant and is tightly integrated with the Standard Template Library (STL) via STL-compliant stream iterators, and natively supports the STL std::string's in otl_streams. Further information about the library is available on the website.

7.1.2 The Tables

To store the data that the robot will gather, a suitable database schema was required. Two different kind of schemas were experimented with in the course of this research. These are listed in Tables 7.1 and 7.2.

Table 7.1: The Map Table: based on the `(key, p)` schema

Field Name	**Field Type**
Key	NUMBER(10) PRIMARY KEY
P	NUMBER(11,10)

Table 7.2: The Map Table: based on the `(x, y, p)` schema

Field Name	**Field Type**
X	NUMBER(10)
Y	NUMBER(10)
P	NUMBER(11,10)

In Table 7.1, the key is a generated value for every unique combination of the robot's *x* and *y* Cartesian co-ordinates. It is generated by first left-shifting *x* by 16 bits and then adding *y* to this new value. Following the reverse procedure, the corresponding *x* and *y* can be retrieved from a given key. In Table 7.2, there is a two-column B-tree Index combining both X and Y columns. There is no need for a key in this schema.

7.1.3 Database Access Functions

The database access functions specific to this research are defined in a header file `db.h`. This file is included by all application modules in this project which require access to the CSS. This file includes the OTL library header. It provides all database-specific functions to the applications, from connecting to retrieving and updating data. The database header file is nearly the same for both the schemas, differing only where the key needs to be sent as a parameter. For the `(key, p)` schema, only the `key` value is sufficient to identify the row of interest, whereas for the `(x, y, p)` schema, both `x` and `y` are required to select the required row.

The functions defined in the database header file are listed below:

1. *connect()*

   This functions creates a global connection to the database. This function is called when the application initializes.

2. *disconnect()*

   This function closes the database connection. This function is called just before the application exits.

3. cleartable()

This is used to clear the map table in the database. It is called at the start of a new mapping session. The execution of this function deletes all rows from the map table.

4. addrows(key, p)

   This function is used to add a new grid cell to the map table. It adds a row with values, `key` and `p`.

5. displayrows()

   This is useful for viewing the contents of the map table. It displays the contents of the map table in a tabular fashion.

6. updatedb(key, p)

   Once a grid has been created, it will be updated often. This function updates a grid having `key` value with a newly computed belief `p`.

7. getdata(key)

   To be able to compute a new value for `p`, the current value must be retrieved. This function returns the belief `p` referred to by `key`. If the supplied key does not exist in the map table, then it signifies that a new grid has been discovered and this function returns a special value P_IF_NULL which is detected by the mapper before performing an update.

8. commit()

   This function send a `commit` command to the database. It is called every after 300 updates and after every 300 inserts.

9. generatekey(x, y)

   Given a value for `x` and `y`, this functions generates the corresponding `key` which will be used to identify the unique `x` and `y` combination. This function is only useful when dealing with the `(key, p)` schema, and does not exist in the header file for the `(x, y, p)` schema.

10. getxyfromkey(*x, *y, key)

    Given a value for `key`, this functions returns the corresponding `x` and `y` which is the kind of data that the mapper and localizer can understand. This function is only useful when dealing with the `(key, p)` schema, and does not exist in the header file for the `(x, y, p)` schema.

11. getGWidthGHeight(*GWidth, *GHeight)

    This functions is used by the localizer and the *pgmcreator* application to retrieve the height and width of the dynamically generated map of the environment.

7.2 Pgm Creator

This application is used to generate a pgm image file from the map stored in the CSS. It is very useful for viewing the results of the mapper, and for confirming the data that the localizer expects. This application can be executed even if the mapping is not complete, to view how the map is being generated. The dynamic nature of this application does not require knowledge of the height and width of the image to be created: it automatically gets that information from the amount of data that has been stored in the map.

7.3 Mapping

The mapping algorithm processes data collected by a wanderer program. This wandering may be incorporated into the mapper itself, but breaking up the code into distinct modules, is a more efficient and simpler approach. The wanderer continuously sends commands to the robot to make it wander about the environment without colliding with any obstacle in its path. A significant degree of randomness has been introduced in the wanderer to allow the robot to explore as much of the world as possible in the shortest time. As the robot moves, it gathers information about its environment and outputs that data to a file in the form:

$[\,x,\ y,\ \theta,\ 16\ \mathit{sonar\ readings}\ldots\,]$

each line delimited by a carriage return. Once wandering is complete and a sufficient amount of data has been collected, the mapper starts up and begins to read this data file. Therefore, the information gathered by the wanderer is used to create the map. The algorithm used to create this map is shown in Figure 7.1. The *processData* function is called on each and every line of data stored by the wanderer.

7.4 Monte Carlo Localization

Monte Carlo Localization was carried out using the Centralized Storage System to implement DEOGs. The algorithm is shown in Figure 7.3. The component diagram for the complete application is shown in Figure 7.2. The *w*anderer program is initially run to gather data about the environment so that the robot can make a map from it. This application causes the robot to explore its environment and output all the raw sonar data

```
processData(dataset):
    $p\_in\_table \leftarrow 0$
    $gx \leftarrow dataset[0]$
    $gy \leftarrow dataset[1]$
    $\theta \leftarrow dataset[2]$
    for each sonar
        read sonar reading from data file
        for each point $(x, y)$ under the sonar's scan
            calculate $pemp \leftarrow$ Bayesian Sonar Model
            $key \leftarrow generatekey(x, y)$
            $p\_in\_table \leftarrow getdata(key)$
            if $p\_in\_table \equiv P\_IF\_NULL$
                $oldpemp \leftarrow 0.5$
                $PsEPE \leftarrow pemp * oldpemp$
                $pemp \leftarrow PsEPE * 1.0/((1 - pemp) * (1 - oldpemp) + PsEPE)$
                $addrows(key, pemp)$
            else
                $oldpemp \leftarrow p\_in\_table$
                $PsEPE \leftarrow pemp * oldpemp$
                $pemp \leftarrow PsEPE * 1.0/((1 - pemp) * (1 - oldpemp) + PsEPE)$
                if $oldpemp \neq pemp$
                $updatedb(key, pemp)$
```

Figure 7.1: The DEOG Mapping Algorithm

collected. This raw sonar data from the wanderer is then used by the *m*apper to create an occupancy grid map of the environment. Since it is using DEOGs, the mapper does not need to know the size of the environment being mapped and dynamically generates a map. This map is stored in the *C*SS, which allows the map to expand and contract as per the environment. Once the map has been created it is used by the *l*ocalizer to localize the robot. Most practical implementations would have some sort of path-planning and goal-oriented navigation also added to this application. For this research, to test localization we let the robot wander around in the environment and test the actual position of the robot returned by the simulator with the location deduced by the localization algorithm. By measuring the mean error and standard deviation between the value returned by the MCL algorithm and the actual value of the robot's position returned by the simulator, we deduce the performance of the algorithm.

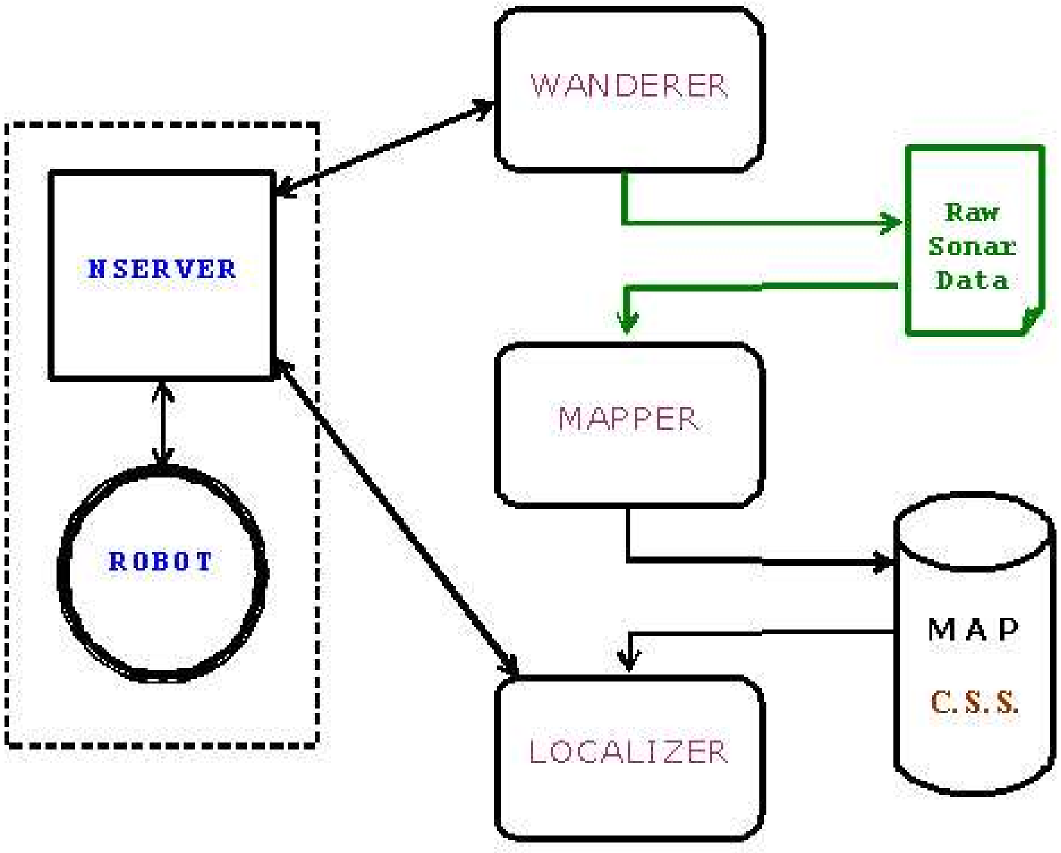


Figure 7.2: Component Diagram

Presented here are the results from two of the environments the experiment was run on. Map1 is an environment which has somewhat similar objects, whereas Map2 is more asymmetrical than Map1. In all cases, it was found that once MCL localized onto the robot, it would not lose track of the robot unless the robot was forcibly moved to another location from its currently localized one. The screenshots are of the experiment running on the simulator. The colored dots are the samples that MCL picked out as probable locations of the robot. Initially we see a lot of blue colored samples, which represent the most probable locations after the first iteration of the algorithm (see Figure 7.3). As the algorithm receives more sensor data, it is able to reduce the most probable samples to only one or two that have the maximum weight in the whole sample set. In all the runs, the actual encoder position of the robot was retrieved from the simulator. This position was stored along with the result given out by the MCL() algorithm for the same position of the robot. Both the locations were compared and Mean Errors as well as the Standard Deviation were computed. These results are listed below. A plot of the results was also generated which shows how closely the algorithm is able to follow the actual position of the robot. The actual position of the robot is represented by the red line and the green lines represents the MCL() position.

```
voidMCL():
    ConnectToCSS
    SizeOfMap $\leftarrow$ Read Map Table
    Map $\leftarrow$ Map Table
    SampleSetSize $\leftarrow$ SizeOfMap * NoOfOrientations
    InitialWeight $\leftarrow$ 1.0 / SampleSetSize
    Initialize SampleSet(SampleSetSize) $\leftarrow$ InitialWeight
    while(1)
        $a \leftarrow$ Action Taken
        $o \leftarrow$ Sonar Reading
        $s \leftarrow$ Random Samples, Weight[s] $>$ threshold
        $s' \leftarrow p(a,s)$
        Update Weights $\leftarrow p(o,s')$
        Output Location $s'$, Max Weight[s’]
```

Figure 7.3: The DEOG Monte Carlo Localization Algorithm

In Screenshot 1 (Figure 7.4 & Figure 7.5), the initial position of the robot is the center of the map, facing the bottom-right corner of the ”L”-shaped obstacle. Therefore all initial samples correspond to locations which would give the same kind of sensor reading. As the robot moves a little, the sensor reading changes and the algorithm is able to localize the robot to its correct position. This is shown by the different colored dots moving with the robot, and represent the path taken by the robot. The Mean Error and Standard Deviation for this run are shown in Table 7.3. We can see that the localizer is able to correctly deduce the location of the robot. Once it has deduced this position, it updates the motion model according to the motion of the robot. It also continues to receive sensor readings which further amplify the weight of the deduced location.

The other experiment we consider was run on Map2 itself, but from a different starting location. In this run also, we see that the localizer is at first not able to detect the correct location of the robot. Initially it gives equal weightage to other locations from which positions, the robot could have received a similar range reading. But as the robot moves, the localizer is able to locate the robot and henceforth move with it. The robot is only using the MCL algorithm with DEOGs to deduce its current location. The results for this experiment are shown in Table 7.4. The corresponding screenshot and the plot are shown in Figures 7.6 and 7.7 respectively.

The third run we consider was run on Map1. This is a slightly symmetrical map, with the objects looking nearly the same. Its observed that the algorithm is trying many different probable positions that the robot could correctly be at. When the robot is forcibly moved from its location many times to another one, to test the strength of the localizer, the localizer would start picking up samples randomly once again, until it found the correct location of the robot. In just a few more iterations, it is able to correctly locate the robot. Once the localizer has correctly identified the location of the robot, its belief improves with every sensor reading, and the localizer then maintains that location of the robot as the correct location, thereby recovering from the kidnapping.

Table 7.3: Run 1: Map2 Error in Localization

	Mean Error	Standard Deviation
X Co-ordinate	8.59055 inches	1.19190 inches
Y Co-ordinate	5.22283 inches	3.49318 inches
Theta	0.0 degrees	0.0 degrees

Table 7.4: Run 2: Map2 Error in Localization

	Mean Error	Standard Deviation
X Co-ordinate	3.82836 inches	3.44103 inches
Y Co-ordinate	8.15522 inches	1.01169 inches
Theta	0.0 degrees	0.0 degrees

Table 7.5: Run 3: Map1 Error in Localization

	Mean Error	Standard Deviation
X Co-ordinate	7.73698 inches	1.57169 inches
Y Co-ordinate	3.57057 inches	2.83530 inches
Theta	0.0 degrees	0.0 degrees

Table 7.6: Run 4: Map1 Error in Localization

	Mean Error	Standard Deviation
X Co-ordinate	4.18384 inches	2.23462 inches
Y Co-ordinate	4.13844 inches	2.51715 inches
Theta	3.81058 degrees	11.0300 degrees

Table 7.7: Run 5: Map2 Error in Localization

	Mean Error	Standard Deviation
X Co-ordinate	9.27500 inches	0.31001 inches
Y Co-ordinate	8.17859 inches	1.01169 inches
Theta	0.0 degrees	0.0 degrees

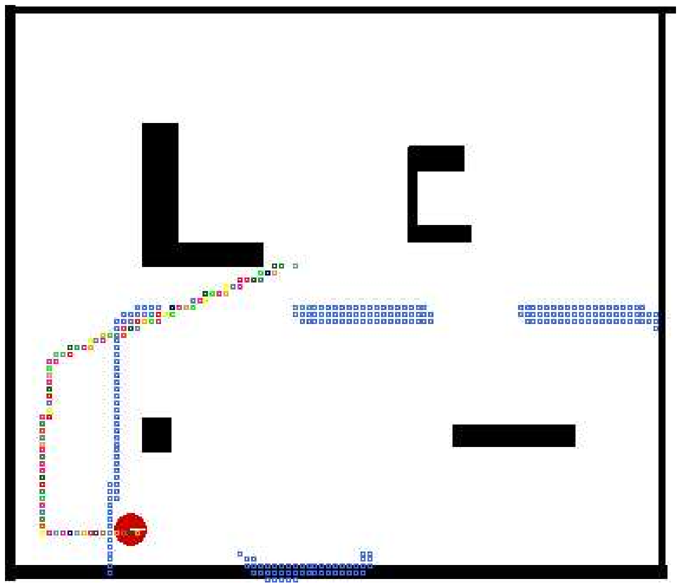


Figure 7.4: Screenshot 1: MCL on Map2

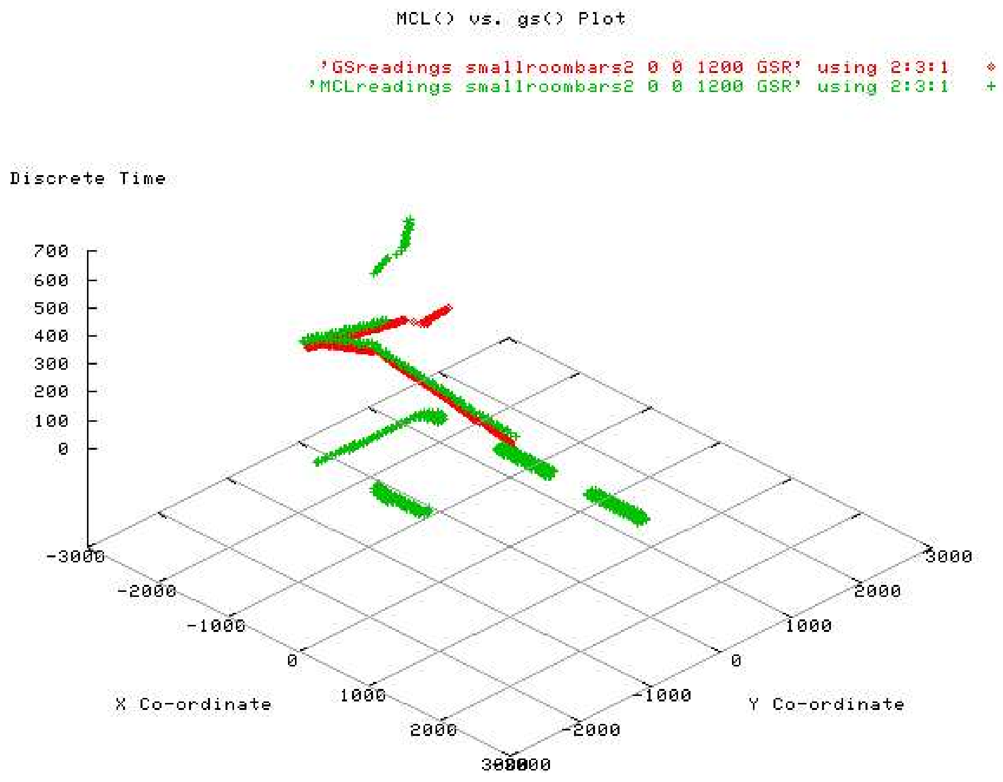


Figure 7.5: Plot 1: MCL vs. gs() on Map2

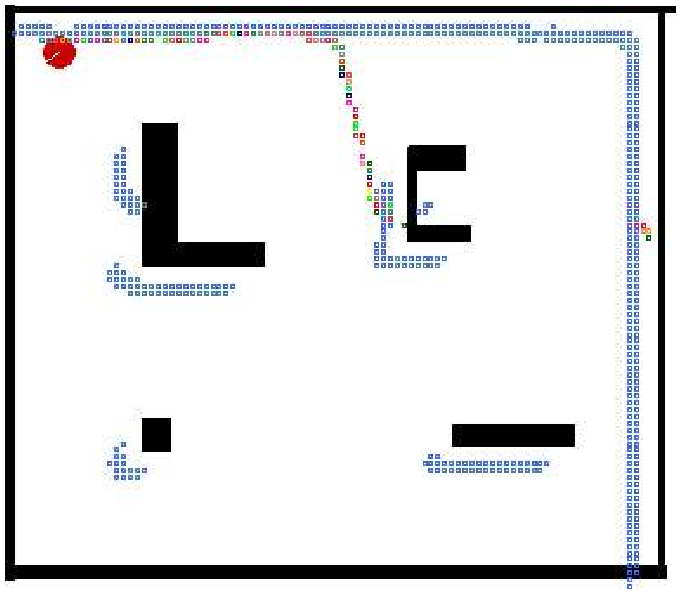


Figure 7.6: Screenshot 2: MCL on Map2

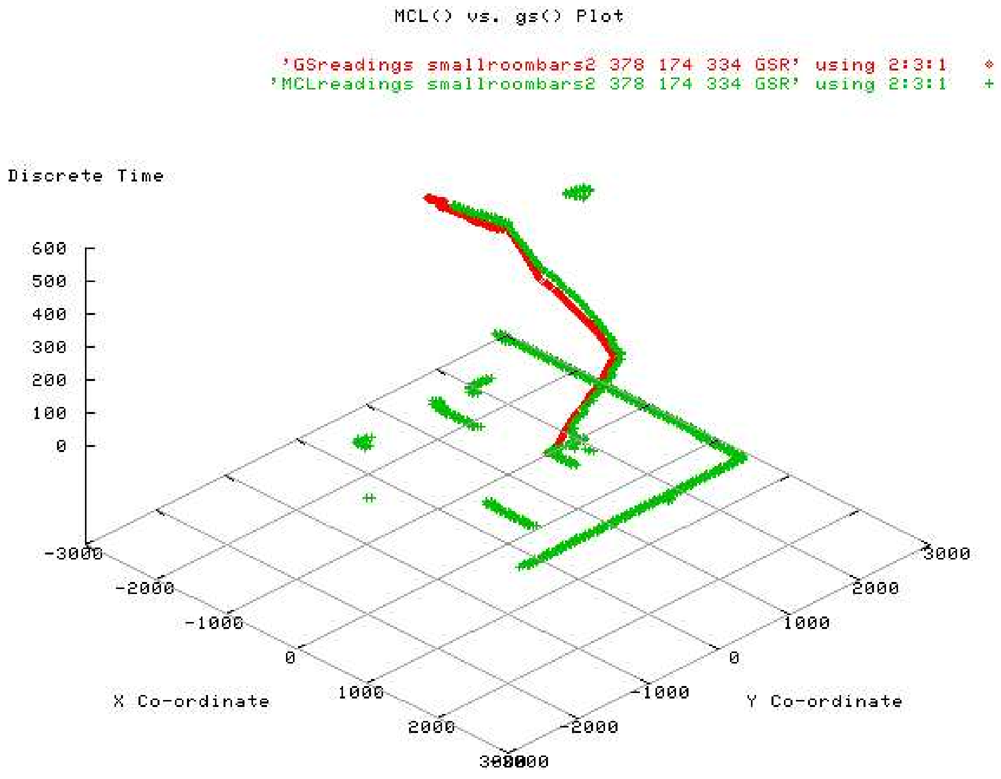


Figure 7.7: Plot 2: MCL vs. gs() on Map2

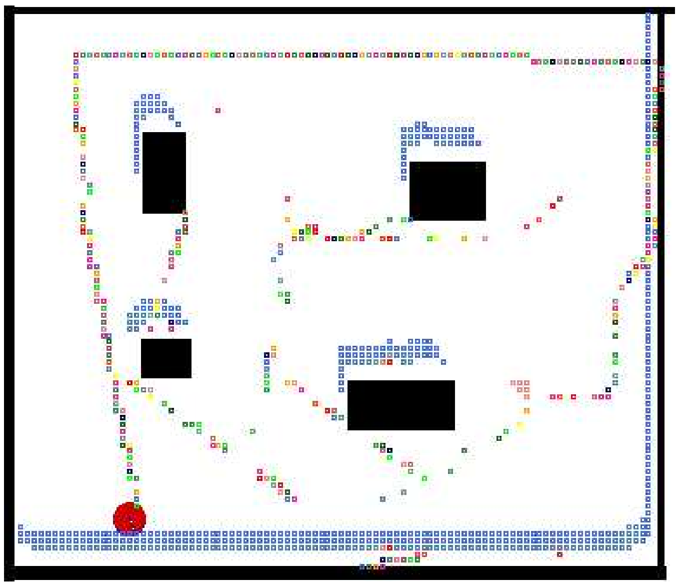


Figure 7.8: Screenshot 3: MCL on Map1

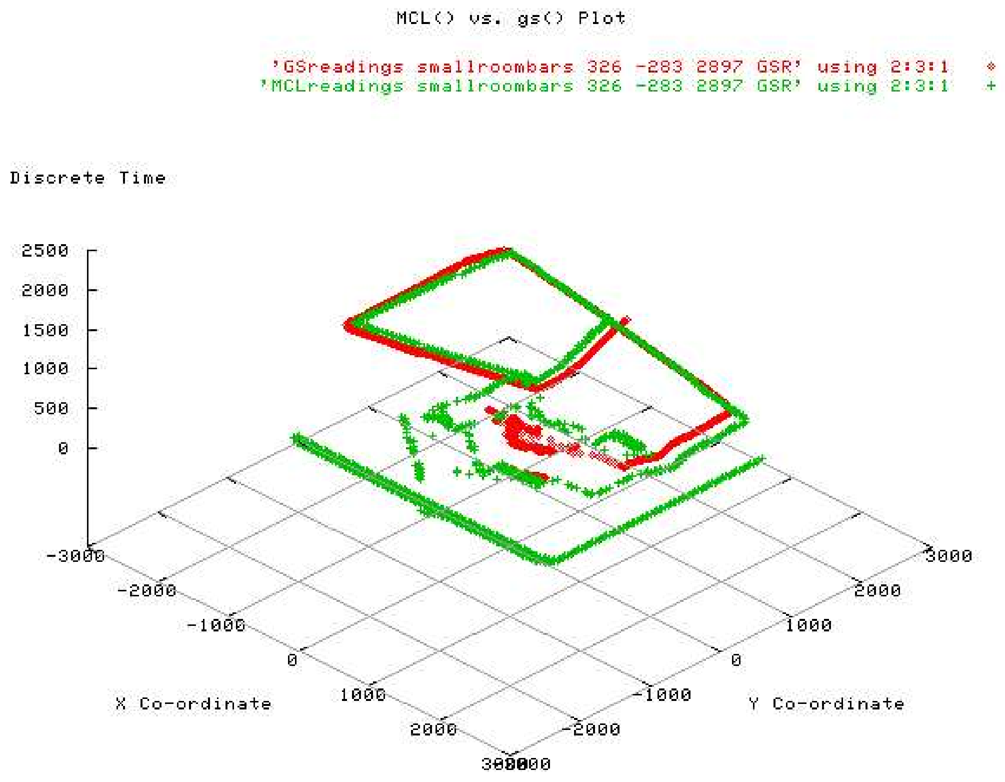


Figure 7.9: Plot 3: MCL vs. gs() on Map1

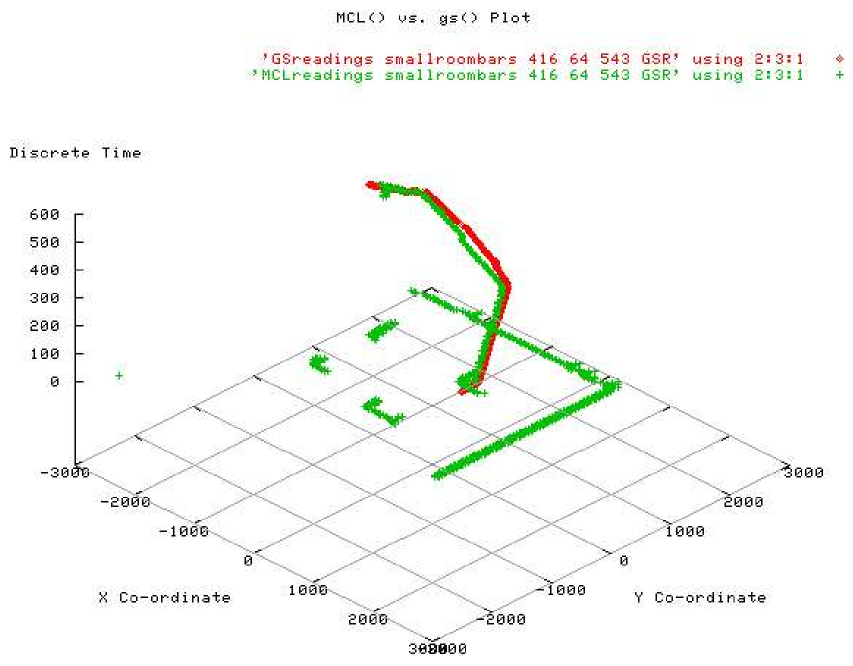


Figure 7.10: Plot 4: MCL vs. gs() on Map1

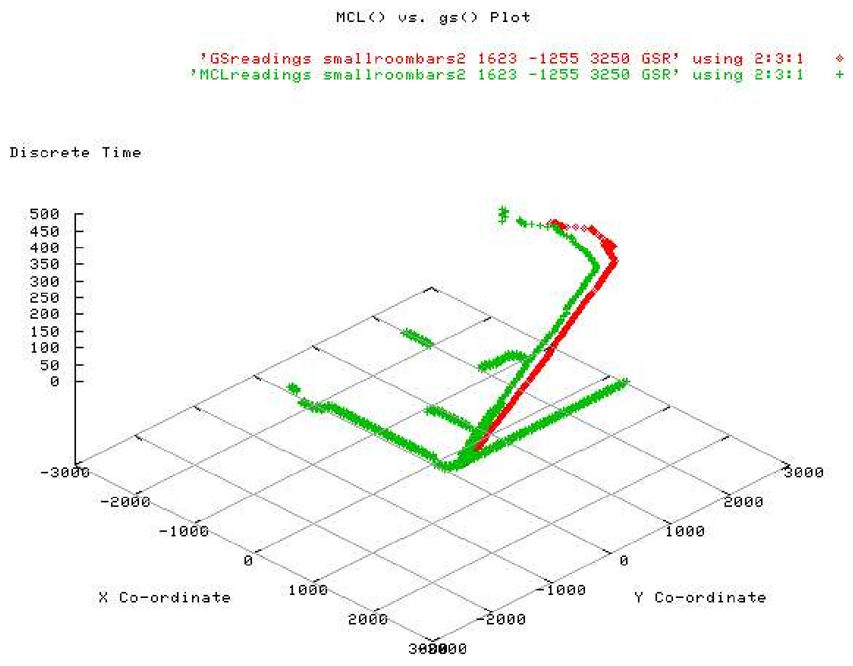


Figure 7.11: Plot 5: MCL vs. gs() on Map2

CHAPTER 8

CONCLUSIONS

This research has achieved its goal of demonstrating a successful implementation of Monte Carlo Localization on robots using Dynamically Expanding Occupancy Grids and a Centralized Storage System. Most limitations of standard occupancy grids can now be overcome and this research can be used to develop a complete navigational system for mobile robots. Utilizing this approach, several practical applications can be built without any concern about running out of memory because of the size of the environment.

8.1 Limitations

The scope of this research is limited to mapping using Occupancy Grids. Further, the results have been tested in, but are not limited to, ultrasonic range sensors. Many different sensors such as lasers, infrared and vision have not been tested, but apart from modifying the basic sensor model, no further changes to the approach will be necessary.

8.2 Future Work

It would be very interesting to see how well the system copes with the rigorous computation demands of SLAM (Simultaneous Localization and Mapping). This dynamic and central storage model needs to be tested for path-planning and navigational systems to build a completely dynamic infrastructure. Another expansions of this research would be to extend the principles to Dempster-Shafer beliefs.

REFERENCES

Arkin, R. C. (1989, August). Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 92–112.

Balch, T. (1996). Grid based navigation for mobile robots.

Bansal, A. (2002). Monte carlo localization for mobile robots in dynamic environments. Master's thesis, Texas Tech University.

Barnes, L. E., T. Quasny, R. Garcia, and L. D. Pyeatt (2004, June 6-9). Multi-agent mapping using dynamic allocation utilizing a centralized storage system. In *12th Annual Mediterranean Conference on Control and Automation*, Kusadasi, Aydin, Turkey.

Burgard, W., D. Fox, and D. Henning (1997). Fast grid-based position tracking for mobile robots. In *KI - Kunstliche Intelligenz*, pp. 289–300.

Burgard, W., D. Fox, H. Jans, C. Matenar, and S. Thrun (1999). Sonar-based mapping with mobile robots using em. In *16th International Conf. on Machine Learning*, San Francisco, CA, pp. 67–76. Morgan Kaufmann.

Cahut, L., K. Valavanis, and H. Delias (1998, May 16-20). Sonar resolution-based environment mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Volume 3, Leuven, Belgium, pp. 2541–2547.

Chong, K. and L. Kleeman (1999). Mobile robot map building from an advanced sonar array and accurate odometry.

Dedieu, E. and J. del R. Millain (1998). Efficient occupancy grids for variable resolution map building. In *6th International Symposium on Intelligent Robotic Systems*, pp. 195–203.

Dellaert, F., D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. URL: `http://citeseer.ist.psu.edu/35242.html`.

Elfes, A. (1989a). *A probabilistic framework for robot perception and navigation*. Ph. D. thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University.

Elfes, A. (1989b). Using occupancy grids for mobile robot perception and navigation. *IEEE Computer Magazine, special issue on Autonomous Intelligent Machines 22*(6), 46–58.

Ellore, B. K. (2002). Dynamically expanding occupancy grids. Master's thesis, Texas Tech University.

Fox, D., W. Burgard, F. Dellaert, and S. Thrun (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI/IAAI*, pp. 343–349.

Fox, D., S. Thrun, W. Burgard, and F. Dellaert (2001). Particle filters for mobile robot localization. In A. Doucet, N. de Freitas, and N. Gordon (Eds.), *Sequential Monte Carlo Methods in Practice*, New York. Springer.

Howard, A. and L. Kitchen (1996). Generating sonar maps in highly specular environments. In *Fourth International Conference on Control, Automation, Robotics, and Vision*, pp. 1870–1874.

Howard, A. and L. Kitchen (1997, March). Sonar mapping for mobile robots. Technical Report 96/34, Department of Computer Science, University of Melbourne.

J. Borenstein, Y. K. (August 1991). Histogrammic in-motion mapping for mobile robot obstacle avoidance. *IEEE Transactions on Robotics and Automation 7*(4), 535–539.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transactions of the American Society of Mechanical Engineers* (82), 35 – 45.

Konolige, K. (1997, October). Improved occupancy grids for map building. *Autonomous Robots 4*(4), 351 – 367.

Koren, Y. and J. Borenstein (1991). Potential eld methods and their inherent limitations for mobile robot navigation. In *IEEE Int. Conference on Robotics and Automation*, pp. 1398–1404.

Kuchin, S. Oracle, odbc and db2-cli template library, version 4.0. URL: `http://otl.sourceforge.net/`.

Lanthier, M., D.Nussbaum, and A.Sheng (2004, August). Improving vision-based maps by using sonar and infrared data. submitted to Robotics and Applications, IASTED 2004.

Martin, M. and H. Moravec (1996, March). Robot evidence grids. Technical report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Moravec, H. (1988). Sensor fusion in certainty grids for mobile robots. *AI Magazine 9*(2), 61–74.

Moravec, H. and M. Blackwell (1992, September). Learning sensor models for evidence grids. Robotics Institute Research Review.

Moravec, H. and A. E. Elfes (1985, March). High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pp. 116–121.

Moravec, H. P. (1996, September). Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical report, Carnegie Mellon Univerisity.

Murphy, R. R. (2000). *Introduction to AI Robotics*. Cambridge, MA.: MIT Press.

Nomadic Technologies Inc. (1999, July 12). *Nomad Scout User's Manual* (2.7 ed.). Mountain View, CA 94043: Nomadic Technologies Inc.

Olson, C. F. (1999). Subpixel localization and uncertainity estimation using occupancy grids. In *IEEE International Conference on Robotics and Automation*.

Schiele, B. and J. L. Crowley (May 1994). A comparison of position estimation techniques using occupancy grids. In *IEEE International Conference on Robotics and Automation*.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence 99*(1), 21–71.

Thrun, S. (2002). Robotic mapping: A survey. In G. Lakemeyer and B. Nebel (Eds.), *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann. to appear.

Thrun, S., D. Fox, W. Burgard, and F. Dellaert (2001). Robust monte carlo localization for mobile robots. *Artificial Intelligence 128*(1-2), 99–141.

Wallner, F. and R. Dillmann (1994). Efficient mapping of dynamic environment by use of sonar and active stereo-vision. In *Symp. on Intelligent Robotic Systems*, Grenoble, France.

Yamauchi, B. and R. Beer (1998, June 1996). Spatial learning for navigation in dynamic environments. In *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, Special Issue on Learning Autonomous Robot*, Volume 26, pp. 496–505.

Yamauchi, B., A. Schultz, and W. Adams. Integrating exploration and localization for mobile robots. *Autonomous Robots, Special Issue on Learning in Autonomous Robots*.

Youngblood, G., L. Holder, and D. C (2000). A framework for autonomous mobile robot exploration and map learning through the use of place-centric occupancy grids. In *ICML Workshop on Machine Learning of Spatial Knowledge*.