DP-SLAM

by

Austin Eliazar

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Pankaj Agarwal

_____
Larry Carin

_____
Carlo Tomasi

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2005

ABSTRACT

DP-SLAM

by

Austin Eliazar

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Pankaj Agarwal

_____
Larry Carin

_____
Carlo Tomasi

_____

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University

2005

Abstract

We present a novel, laser range finder based algorithm for simultaneous localization and mapping (SLAM) for mobile robots. SLAM addresses the problem of constructing an accurate map in real time despite imperfect information about the robot's trajectory through the environment. Unlike other approaches that assume predetermined landmarks (and must deal with a resulting data-association problem) our algorithm uses the sensor range data directly to build metric occupancy maps. Our algorithm uses a particle filter to represent *both* robot poses and possible map configurations. By using a new map representation, which we call distributed particle (DP) mapping, we are able to maintain and update hundreds of candidate maps and robot poses efficiently. Through careful implementation, we are able to achieve a time complexity which is linear in both the area observed and the number of particles used. Our technique contains essentially no assumptions about the environment yet it is accurate enough to close loops over 100m in length with crisp, perpendicular edges on corridors and minimal or no misalignment errors, despite significant noise and ambiguity.

Contents

List of Tables

List of Figures

Acknowledgements

The author gratefully acknowledges the support of the National Science Foundation, the Sloan Foundation, and SAIC for this research. I also thank the NASA Ames Research Center for funding and other resources supporting the initial research into three dimensional SLAM.

The data sets for Acapulco and Edmonton convention centers were obtained from the Robotics Data Set Repository (Radish) [1]. Thanks to Nicholas Roy for providing this data.

My deep appreciation also goes to Ronald Parr, for his advice and guidance throughout much research and many deadlines.

My love and gratitude to my wife, Rachel Eliazar, for her patience and tolerance throughout this all.

Chapter 1

Introduction

Significant strides have been made towards creating a robot capable of performing completely autonomous tasks. The basic tasks of path planning, localization, navigation and environmental manipulation are well understood, and to some limited degree, have been solved. These components form the basic tools for solving higher level tasks, and allow a robot to operate for some time without human oversight or intervention.

However, one component which has been assumed in all of these areas is the existence of a good map of the environment. This map not only needs to be accurate, but also needs to be a fairly complete map of everywhere that the robot has observed so far in its exploration. Furthermore, it needs to be able to incorporate new information immediately, as the robot explores new areas. This problem of tracking the robot's pose and constructing a map in real-time is known as Simultaneous Localization and Mapping, or SLAM.

In recent years, the availability of relatively inexpensive laser range finders and the development of particle filter based algorithms have led to great strides on the problem of robot localization – determining a robot's position given a known map [2]. Initially, the maps used for these methods were painstakingly constructed by hand. However, the accuracy of the laser suggests its use for map-making as well as localization.

SLAM has long been a stumbling block for autonomous robots. What appears on the surface to be two separate challenges, localization and mapping, are in fact intricately intertwined problems. For the robot to update the map correctly, it is necessary to know where the robot is when it makes an observation. However, when tracking the robot's pose, it is essential to have a good map against which to compare the observations. Solving both of these problems incrementally and at the same time means that a small error in one

solution can easily corrupt all future estimations. In fact, it is the quick accumulation of many tiny mistakes which is commonly the cause of failure for nearly all SLAM methods.

External sensors, such as a global positioning system (GPS), are unsuited for tracking the robot's pose. GPS relies on receiving signals from satellites in orbit, and are easily obscured by trees, buildings, canyons, or other obstacles. Also, GPS coverage is unreliable in many terrestrial locations, and would not be possible for extraterrestrial applications. Finally, the readings provided by GPS are only a latitude and a longitude; there are other important elements of the robots pose, including facing and height displacement, which are unable to be measured by GPS.

The Expectation-Maximization (EM) algorithm provides a very principled approach to the problem of mapping, but it involves an expensive off-line alignment phase [3]. There exist heuristic approaches to this problem that fall short of full EM, but they do not provide a complete solution and they require additional passes over the sensor data [4]. Scan matching can produce good maps [5, 6] from laser range finder data, but such approaches typically must explicitly look for loops and require additional effort to close them. In varying degrees, these approaches can be viewed as partially separating the localization and mapping components of SLAM.

Recent approaches to SLAM have shown some significant progress towards solving the combined, real-time problem. One of the more popular of these methods is FastSLAM, which typifies a group of approaches called landmark-based SLAM. For these algorithms, the map is represented by a set of distinctive landmarks, and a Kalman filter is maintained over these landmark positions. Another group of approaches concentrates more on the underlying topological map, and then builds a more comprehensive map on top of the topology [7].

However, it is important to keep in mind the purpose of the map. Landmarks and topology are fine tools for localization and even simple navigation, but are not very informative

for other applications. To make well reasoned, intelligent decisions about the environment, a more complete picture is needed. To achieve any useful level of autonomy, even to merely travel unaided from one specific point to another, the robot needs to have a fairly complete picture of the world. With this in mind, we concentrate on developing a SLAM method which can produce a dense metric representation of the world, using occupancy grids.

1.1 Objective

The primary goal of this project is to produce highly accurate and detailed maps for both 2-D and 3-D environments in real time. These maps should be complete and informative as well as being accurate over large environments, regardless of the robot's path through the environment. Ideally, SLAM should be a passive algorithm, and explicit behaviors, such as traveling in loops should not be a requirement for accurate mapping. To preserve the robot's autonomy, it should be able to achieve these goals with little or no human intervention.

We have made significant progress towards this goal, in the two dimensional case. The development of a novel algorithm, Distributed Particle SLAM (DP-SLAM), allows for the efficient maintenance of multiple map hypotheses in a principled fashion. A carefully constructed map formulation provides complete maps, which are able to represent areas of uncertainty and semi-transparency.

We present a number of advances in effectively managing resources, allowing better concentration in relevant areas of the state space. The effective use of an efficient map representation, when combined with dynamic programming, allows the algorithm to be run in linear time with respect to both the size of the observations and the number of hypotheses being considered. Surprisingly, this provides an asymptotic running time which is identical to pure localization, for a given number of particles.

To maintain an appropriate proposal distribution in the face of changing environmental

and robot parameters, we have also developed a purely autonomous method for calibration of the robot's motion model. Using the SLAM algorithm itself within an Expectation Maximization (EM) framework, we can create much tighter proposal distributions. This, in turn, can greatly improve the distribution of resources within the state space. With these methods, in conjunction with efficient implementation and a well concentrated proposal distribution, we are able to produce very high quality, accurate two dimensional maps at efficient speeds.

Three dimensional mapping, is still an open area of research. However, we have promising initial results on how to represent the vast amount of data needed in an efficient manner, and how to process the wealth of input information in reasonable amounts of time. We are continuing work on promising methods for many of the related problems, such as the development of effective sensors and their associated observation models. We are also continuing to pursue more efficient representation and manipulation of the data, to overcome the data explosion inherent in moving from 2-D to 3-D.

Chapter 2

Previous Work

2.1 Localization Overview

A simpler, but similar problem to SLAM is robotic localization. For pure localization, the robot already has a map of its environment. The goal is to use the robot's sensors to track the robot's pose as it moves through the environment. The sensors can provide indirect observations of the environment, which the robot can then compare with its internal map of the world. Using these comparisons between the expected observations and the actual readings, combined with the robot's belief of its pose in the past, it is possible to determine the robot's current pose.

Successful localization is not a trivial problem in itself. The sensor readings tend to be noisy and sparse, and many areas within the environment appear to be very similar. Therefore, a probabilistic algorithm is required to track the robot's position. This is achieved either through a Kalman Filter, or a more general Particle Filter.

2.1.1 Particle Filter Review

A particle filter is a simulation-based method of tracking a system with a partially observable state. We briefly review particle filters here, but refer the reader to excellent overviews of this topic [8] and its application to robotics [9] for a more complete discussion.

A particle filter maintains a weighted (and normalized) set of sampled states, $S = \{s_1 \dots s_m\}$, called *particles*. At each step, the robot executes a motion u, and makes an observation o (or vector of observations). The particle filter executes the following steps:

1. Samples m new states $S' = \{s'_1 \ldots s'_m\}$ from S with replacement.

2. Propagates each new state through a Markovian transition (or simulation) model: $P(s''|s', u)$. This entails sampling a new state from the conditional distribution over next states given the sampled previous state.

3. Weighs each new state according to a Markovian observation model: $P(o|s'')$

4. Normalizes the weights for the new set of states

Particle filters are easy to implement have been used to track multimodal distributions for many practical problems [8].

2.1.2 Particle Filters for Localization

A particle filter is a natural approach to the localization problem, where the robot pose is the hidden state to be tracked. The state transition is the robot's movement and the observations are the robot's sensor readings, all of which are noisy.

The change of the state over time is handled by a motion model. Usually, the motion indicated by the robot's odometer is taken as the basis for the motion model, as it is a reliable measure of the amount that the wheels have turned. However, odometry is a notoriously inaccurate measure of actual robot motion, even in the best of environments. The slip and shift of the robot's wheels, and unevenness in the terrain can combine to give significant errors which will quickly accumulate (Figure 2.1. A motion model differs across robots and types of terrain, but generally consists of a linear shift, to account for systematic errors and Gaussian noise. Thus, for odometer changes of x, y and θ, a particle filter applies the noise model and obtains, for particle i,

$$x_i = a_x * x + b_x + \mathcal{N}(0, \sigma_x) \quad (2.1)$$

$$y_i = a_y * y + b_y + \mathcal{N}(0, \sigma_y) \quad (2.2)$$

$$\theta_i = a_\theta * \theta + b_\theta + \mathcal{N}(0, \sigma_\theta) \quad (2.3)$$

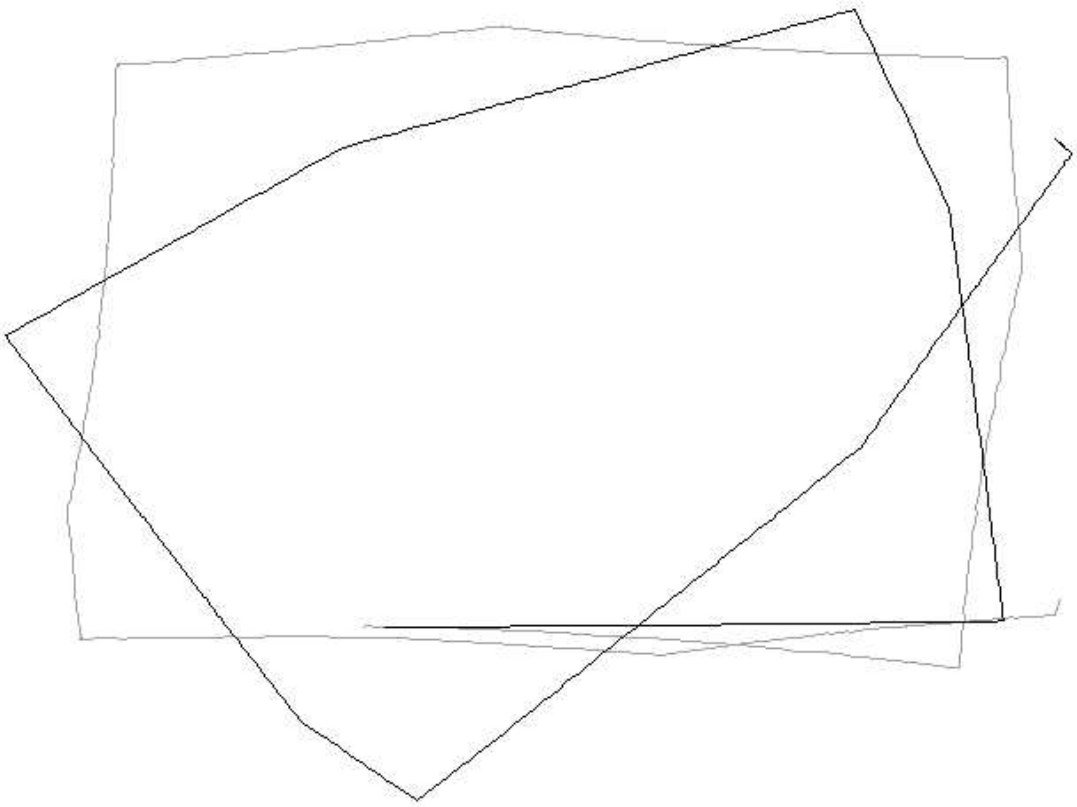

Figure 2.1: A plot of the robot's actual motion, shown in grey, compared with the trajectory described by the odometry, shown in black.

The a and b terms are linear correction to account for consistent errors in motion. The function $\mathcal{N}(0, \sigma)$ returns random noise from a normal distribution with mean 0 and standard deviation σ, which is derived experimentally and may depend upon the magnitudes of x, y, and θ.

After simulation, we need to weight the particles based on the robot's current observations of the environment. For pure localization, the robot has a map stored in memory. The position described by each particle corresponds to a distinct point and orientation within that map. Therefore, it is relatively simple to determine what values the sensors should return, given the pose within the map. The standard assumption is that sensor errors are normally distributed. Thus, if the first obstruction in the map along a line traced by a laser cast is at distance d and the reported distance is d', the probability density of observing discrepancy $\delta = d' - d$, is normally distributed with mean 0. Given the model and pose, each sensor reading is correctly treated as an independent observation [10]. The total posterior for particle i is then

$$P_i = \Pi_k P(\delta_{ik} | s_i, m),$$

where δ_{ik} is the difference between the expected and perceived distances for sensor (laser cast) k and particle i.

Particle filters have been used successfully in many different ways to localize a robot. One of the most notable is the pair of robots used as museum tour guides, Rhino and Minerva, deployed at the Deutsches Museum Bonn and the Smithsonian's National Museum of American History, respectively, for a period of several days at a time [2]. These robots were able to guide visitors to various exhibits successfully, even in the presence of significant noise, most notably from the people themselves.

Since using particle filters in this manner is so successful at localization, it is natural to consider them for the problem of SLAM. However, all previous attempts are stopped at the same point: how to represent the uncertainty in the robot's pose when updating

positions of possible obstructions in the map. The conceptually correct method is to allow each particle to maintain and update its own map. However, implementation of this idea requires extraordinary amounts of memory, and is prohibitively slow, discouraging further research into this approach. We will return to this issue in later sections for a more detailed discussion.

2.2 Landmark SLAM

One popular approach to SLAM is to select distinctive features in the environment to use as landmarks [11, 12]. These landmarks can then serve as navigational guides to determine the robot's pose in the world. The assumption is that these landmarks can be chosen to be distinctive enough that the robot can easily correlate a landmark that is currently observed with one of the previous observations of that same landmark. These landmark-based methods are algorithmically different from other SLAM methods, in that instead of using a particle filter to track the robot pose, they use a Kalman Filter.

We present a brief overview of Kalman filters here, but refer the reader to an excellent presentation of the topic [13] for greater detail and depth. For a complete basic treatment of Kalman filters as applied to SLAM, we refer the reader to the original paper in the field by Cheeseman, Self and Smith [14].

2.2.1 Kalman Filter Overview

A Kalman filter is a well established method tracking the state of a stochastic linear equation $s \in \Re^n$.

$$s' = As + Bu + w,$$

At each discrete time step, a noisy measurement of the state is made, $o \in \Re^m$

$$o = Hs' + v,$$

The v and w terms represent the noise present in the motion and observation equation, respectively. These sources of noise are assumed to be Gaussian and independent of each other.

$$P(w) = N(0, Q), P(v) = N(0, R),$$

Q is called the *process covariance* and R is the *measurement covariance*. Both of these covariance matrices are allowed to vary between time steps.

As well as tracking the state itself, a Kalman filter also maintains the entire covariance matrix of the state estimate P. At each time step, upon observing the measurement o, the Kalman Filter does the following:

1. Predict the next state according to the transition model. Since the noise is assumed to be mean zero, it is ignored for this step. $s' = As + Bu$

2. Update the covariance of the state, thus increasing the uncertainty. $P' = APA^T + Q$

3. Compute the Kalman gain, to determine the amount of strength that the new observation should have in updating the state. $K = P'H^T(HP'H^T + R)^{-1}$

4. Update the state by averaging it with the new observation, weighted by the Kalman gain. $s'' = s' + K(o - Hs')$

5. Update the state covariance, to indicate the increased certainty from the new observation. $P'' = (I - KH)P'$

Kalman filters are commonly used as an exact solution to tracking a wide variety of linear systems. Unfortunately, the system being tracked during localization is distinctly nonlinear, as there exists an inherent trigonometric relation between the lateral and rotational motions of the robot. Therefore, it is necessary to use an Extended Kalman Filter (EKF) to create an approximate solution. In the EKF, we use the first Taylor series expansion of the nonlinear equation around the current state to create a linear approximate to the

system. This means that we need to incorporate the Jacobian matrix into the above steps, giving us a new series of equations.

1. $s' = f(s, u)$
2. $P' = APA^T + WQW^T$
3. $K = P'H^T(HP'H^T + VRV^T)^{-1}$
4. $s'' = s' + K(o - h(s'))$
5. $P'' = (I - KH)P'$

In the above equations, A and W are the Jacobian matrices of partial derivatives of the process function f with respect to the current state and the process noise, respectively. Similarly, H and V are the Jacobians of the measurement function h with respect to the current state and the measurement noise. Naturally, A and H will need to be recomputed round the new state estimate at each time step.

It should be noted that the Extended Kalman Filter is an approximate solution to the inherently nonlinear system. This possible source of error should be kept in mind when considering an EKF for the the problem of LAM. In addition, all distributions are assumed to be Gaussian, and thus unimodal. This assumption should be treated skeptically in the case of tracking the robot's pose.

2.2.2 Kalman Filters for SLAM

One of the major difficulties for using Kalman filters for SLAM is the problem of data association. The observation model assumes that there is a known transformation between the measurements and the objects being modeled. However, most aspects of the environment are very similar, and it is not within the capacity of the sensor to distinguish directly which specific object is being observed at any given time.

This is where the concept of landmarks is used. The assumption is made that the robot can make direct observations of a certain subset of the objects in the environment, and recognize these specific objects unambiguously. Therefore, whenever an observation is made, it is possible to know exactly which object is generating that specific measurement. Once this problem of known data association is solved, the state estimation problem becomes significantly more manageable.

As we have noted before, the state tracked in SLAM is both the robot pose and the map. In the case of landmark-based SLAM algorithms, the map that the robot is using to localize is the set of landmark positions. The maintenance of this type of map fits in very nicely with the Kalman filter framework. Each landmark can be represented by a set of spatial coordinates in the state estimation. This allows the landmarks to have correlated Gaussian distributions over their positions, as expressed in the covariance matrix. The robot's own pose is also included in the state, and is the only portion of the state estimation affected by the state update step of the Kalman filter; the landmarks are all assumed to be stationary in the world.

2.2.3 Variations on Landmark SLAM

The use of a Kalman filter provides a principled, closed form solution to the SLAM problem. However, there exist several problems with this formulation. First, many of the assumptions of Gaussian distribution are inaccurate. In particular, the distribution of robot poses can be distinctly multimodal, due to sensor ambiguity. This problem is especially exacerbated when landmarks are not completely unambiguous. Second, maintaining the covariance matrix between the entire set of landmarks requires $O(n^2)$ running time, where n is the total number of landmarks in the map, in order to update the map. Notice that this complexity has the undesirable quality that it is dependent on the total size of the map. Thus, the running time can quickly become impractical as the robot is deployed over a

large area, even when the landmarks maintain a constant density. Finally, and perhaps most importantly, the map which is maintained is only a map of landmarks; other, less distinctive features in the environment, including walls and similar obstructions, will not be represented. As a result, the map is only useful for localization and high level navigation. Any interaction with the world, including local path planning and obstacle avoidance, requires a more complete map of the environment.

Another limitation of landmark-based methods is the selection and recognition of the landmarks themselves. The problem of allowing the robot to select features of the environment autonomously to use as landmarks is still an open question. Without some form of object recognition, the task of picking an aspect of the world which is distinct from several different points of view, in a manner general enough to be used in a wide variety of environments, can be very difficult. Furthermore, the recognition of landmarks can be a difficult data association problem. Attempting to identify a certain feature uniquely as a specific landmark that has been seen before, particularly when leaving open the possibility that this may be a new, previously unobserved landmark, is a very difficult task. This process is only hindered by the noisy, limited sensors available to the robot.

Finally, the Kalman filter representation is incapable of incorporating negative information into the state estimation. Whenever an observation of a landmark is made, the state can be updated. However, there is no method provided for updating the state when a landmark *should* be observed, according to some portion of the probable state space, but isn't. This negative information is potentially a very important source of information which is completely ignored in landmark based SLAM.

Some of these problems have been addressed by existing work. One of the more prominent methods is the use of thin junction trees to approximate the covariance between landmark position estimates [12]. This treats the covariance as a sparser set of relationships between nearby landmarks. The ability of a single landmark to influence its neighbors is

restricted to allow propagation only if the influence has reached a certain threshold. Therefore, most updates to the map affect only a local set of landmarks, but more informative events, such as the completion of a loop, are still allowed to propagate through the entire map. Therefore, the average running time of the Kalman filter is greatly reduced, while still maintaining a good approximation of the full covariance matrix.

A similar approach uses a sparse extended information filter to approximate the EKF by the most important pairwise constraints [15]. Much like the thin junction trees, this method greatly reduces the amount of information that needs to be updated at each time step, by passing information along the most informative constraints.

The other popular improvement on landmark-based SLAM is FastSLAM [11]. This algorithm uses a Rao-Blackwellized particle filter [16] to track the robot's pose. This allows the map to still be represented as a set of Gaussian distributions of the landmark positions like a Kalman filter, while sampling distinct hypotheses for the robot's pose, in the same manner as a particle filter. Given the robot pose, all of the landmark positions become independent [10]. This important consequence greatly improves the running time, since each landmark can now be treated as a separate Kalman filter, with trivial dimensions. Furthermore, it no longer requires the distribution of robot poses to be constrained by a Gaussian, allowing for more flexible, multimodal distributions. It is important to note, however, that this method of sampling robot poses implies that the maps are now dependent directly on the individual robot pose, and thus each particle needs to maintain its own map of landmark positions, which can lead to other complications for running time and memory. This method has met with considerable success, when the issues of data association are solved, and can produce high quality maps in large environments.

Since the data association problem is such a crucial part of an effective landmark-based SLAM method, there are a number of methods for solving this recognition problem. In one effective method, data associations are always chosen so as to maximize the probability of

the current observation [11]. In different approach, the raw sensor information is composed into a "template" for the landmark. Scan correlation is then used to identify these landmarks [17].

2.3 Hybrid Topological SLAM

The predominant alternative to landmark-based SLAM methods is a loose collection of topologically based methods. This topology can be represented either explicitly, in a graph [7, 18, 19], or detected indirectly, through scan matching [5, 6]. Regardless of the representation, there are a couple of important underlying properties of this variety of algorithms. They all attempt to represent a more complete map than just a set of landmarks, and more importantly, they all attempt to exploit the topology of the environment to achieve consistency, and, hopefully, accuracy.

At the core of these algorithms is a method by which to detect when a loop is closed. That is, they are specifically seeking to detect when the robot's trajectory doubles back on itself. The proper alignment of these loop closing events gives much the same information that is captured by the Kalman filter when a landmark based method closes a loop, and the entire trajectory between starting at that point and returning to it can be refined by this extra information. Therefore, these methods attempt to leverage the topological information explicitly, while still maintaining a more complete map of the environment, and avoiding the data association difficulties inherent in landmark based methods.

However, loop closing is a much more difficult problem without the use of explicit landmarks. Difficult questions arise of how to detect such an event, and how to align the two perspectives on the same scene properly. Without explicit landmarks, data association is ambiguous, and both detection of a closed loop and alignment of the two pieces can lead to errors and false positives. Furthermore, exactly how to propagate this new information is a difficult question. Correcting the intervening trajectory is difficult to accomplish in a

principled fashion, without a set of covariances to use. Lacking any method of enforcing accuracy, the correction is instead usually performed in a manner emphasizing consistency. While not widely adopted, a method has been proposed for modeling the residual errors [20]. These residuals could then be used to distribute the error detected by the loop closure.

The major drawback of these methods is their failure to be truly "simultaneous". Detecting loop closures, and even more so, applying the additional information supplied by them, is a very computationally intensive process, which is likely to cause a significant discontinuity in the robot's ability to process the algorithm in real time. Furthermore, the quality of the maps are highly variable. Even though the final maps which are produced can look very good, they are still not very accurate, as the corrections to the trajectory are not necessarily applied to the correct portions. Resultant maps often have bends, elongations, or other distortions from the correct map, as the loop closing event attempts to achieve consistency in any way possible. Perhaps more importantly, even these consistent maps can only be produced at distinct intervals. In the time between loop closures, the maps can degrade arbitrarily.

Chapter 3

Map Representation

3.1 Occupancy Grids

Occupancy grids, also known as evidence grids, are a popular method for describing the parameters of an environment, as they are very intuitive method for representing the world. Occupancy grids were originally developed at Carnegie Mellon University in 1983 for sonar navigation in the lab [21, 22]. They quickly became popular due to their ease of use with sensor fusion and robot navigation [23].

Envision the entire world divided up into a regular grid of squares, all of equal size. Each of these grid squares corresponds to a physical area in the world, and as such, each square contains a different set of objects or portions of an object. An occupancy grid is an abstract representation of these sections of the world, containing information on whether that square in the real world is occupied. This notion of occupancy can change depending on the application, but generally either indicates whether the area would block the passage of a robot, or whether the area would show up on the robot's sensors. Ideally these two concepts should be one and the same, but due to noisy sensors, "invisible" objects, "insubstantial" objects, and many areas only being partially occupied, they sometimes diverge.

Nearly all early localization methods were developed for occupancy grids. Similarly, many path planning algorithms assume either an occupancy grid or a similar dense metric map. This lasting popularity is largely due to the simplicity of the representation, and the detail which is easily available at different resolutions.

3.2 Deterministic and Stochastic Occupancy Models

An occupancy grid can be represented in many different ways. It is important to ensure that the representation is well tailored to both the sensor being used and the application. Most of the standard approaches can be generalized into two types: deterministic and stochastic.

Deterministic occupancy grids are the simplest implementation, and have a discrete set of values for the grid squares. These typically are EMPTY and OCCUPIED, and sometimes include a value for UNKNOWN or UNOBSERVED. This clear and simple approach tends to be popular for maps of the environment created by hand or for simulated environments, for use with path planning algorithms, or other methods having to do with physical interaction with the world. They give a very clear cut description of the physical occupancy of the world from a spatial standpoint. However, they tend to be an over simplification for sensors, since very rarely will the sensor ever see square patch of the world which is both completely occupied and always accurately observed. For this reason, stochastic maps are much more popular for localization, mapping, and other methods which rely on an observational interaction with the world.

Stochastic maps have a notion of OCCUPIED and EMPTY, just like deterministic maps, but instead of viewing these values as absolutes, they have a sliding scale of various degrees of occupancy. These values are affected by a variety of factors, including what percentage of the square is believed to be occupied, and how transparent the object is to the sensor. Since these numbers represent the behavior of a specific sensor with the given area, it makes sense that the method for representing this uncertainty would change for different sensors. Both the stochastic representation and the corresponding observation model need to be tuned properly for the device used.

3.3 Laser Model

The recent development of reasonably priced laser range finders has quickly made them a dominant sensor for use in mapping as well as localization. Unlike other sensors, laser range finders can give accurate measurements within a few centimeters, and are effective out to ranges of 20 meters or more. Another important difference between laser range finders and sensors such as sonar is that the laser traces a very thin line through the world, as compared to the wide cone from sonar. As such, the laser can provide much more precise and accurate readings, compared to previous sensors.

There are three main sources of uncertainty that we would like to accommodate in our model. The first has to do with small objects, and irregular surfaces. Regardless of the resolution of our grid, there will always be objects which do not fit well within a grid square. Rough surfaces, such as rocks and plants, as well as small items like fences and bushes are very difficult to represent explicitly in a deterministic fashion. Another issue arises from the discretization of the world. Most surfaces do not align well with a given grid, since they do not happen to be axis-parallel. Instead, they will only partially occupy certain squares, and they will give differing readings depending on which portion of the grid square is scanned. This can be particularly problematic for surfaces which are nearly parallel to the current scan. Lastly, there are objects in the world that behave in manners too complex to easily model, such as moving people, or surfaces which can occasionally reflect the sensor. We would like a method which can recover gracefully from these aberrant events, instead of placing permanent errors in the map.

The idea of using probabilistic map representations is possibly as old as the topic robotic mapping itself [21]. Many of the earliest SLAM methods employed probabilistic occupancy grids, which were especially useful for sonar sensors prone to noisy and/or spurious measurements. However, by concentrating on a model for a laser range finder, and the behavior of our own algorithm, we can develop a more appropriate method for

representing uncertainty in the map, which takes into account the distance the laser travels through each grid square.

The main challenges for devising a probabilistic map representation are in developing a representation of the environment that is consistent, flexible, and easily updated. Our basic assumption is that each grid square responds to laser light in a manner that is independent of neighboring squares. Moreover, the behavior is independent of the angle of incidence or the part of square which is struck; the behavior with respect to a particular square depends only on the distance the laser travels through the square.

Our assumptions are, of course, false for most environments. However, they are a plausible relaxation of the rather strict assumptions made in the deterministic map. Intuitively, our model corresponds to the assumption that between scans, the obstructions in each grid square are redistributed uniformly within the square. Obviously, more sophisticated approaches that retain more information are possible and some are under consideration for future work. However, if the robot is not permitted enough memory to recall the precise locations of objects within a grid square (or if the laser does not have enough accuracy to make such records worthwhile), the assumption of uniformity is quite natural.

3.4 Map Representation and Observation Model

Since maps are already fairly large data structures, we would like to maintain a small and constant amount of additional information for each grid square that summarizes the properties of the square. With this restriction, it seems natural to avoid overly complex models by assuming that the probability of stopping at any particular point within the square is uniform over the square. From that point, the specific method of developing these probabilities can be tailored to the behavior of the sensor.

One important goal of our model is the idea that the probability of laser penetration should depend on the distance traveled through a grid square. Earlier approaches to esti-

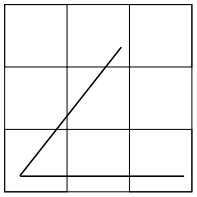

Figure 3.1: Effect of angle on number of grid cells penetrated.

mating the the total probability of the scan would trace the scan through the map, and weigh the measurement error associated with each potential obstacle by the probability that the scan has actually reached the obstacle [24]. In an occupancy grid with partial occupancy, each cell is a potential obstacle.

Consider Figure 3.1, where we can see two possible scans of equal length, originating from the same position, but with different orientations. Note that the axis parallel scan passes through three grid squares, while the diagonal scan passes through 4. (In general, the number of squares visited can differ by a factor of $\sqrt{2}$.) Without a method of weighting the visited grid squares based on the distance the laser has traveled through them, two scans of equal length traveling through similar grid squares can have vastly different probabilities (since they represent a different number of possible laser obstacles) based solely on the orientation of the map. Notice also that some of the squares that the angled scan passes through are only clipped at the corner, resulting in a very short distance spent in those squares. Even if these squares are known to interrupt laser scans with high probability, the effects of these squares on the the total probability of the scan should be discounted. This is largely due to the fact that the sensor only detects the boundaries of objects, and thus map squares which are likely to be occupied are almost always only partially occupied. These effects can cause a localization algorithm to prefer to align some scans along axes for spurious reasons. While this issue has been recognized some existing work [2], is often ignored in other stochastic models, despite being a very important consideration in minimizing the effects of discretization in the map.

Our second goal in developing a laser penetration model was that the the model should

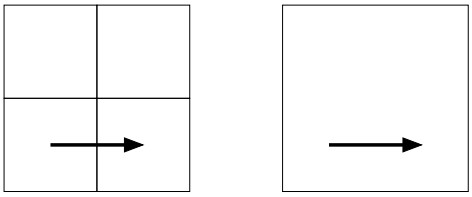

Figure 3.2: Effect of grid resolution on scan probabilities. If the grid squares all have the same density to the sensor, the scan on the left should have the same probability as the one on the right.

be *consistent*. We derive our notion of consistency from the following thought experiment. Consider a laser cast that passes through two adjacent squares of identical composition and travels the same distance through each square (Figure 3.2). The probability that the laser beam will be interrupted should be the same as if the laser had traveled twice the distance through a single, larger square of identical composition to the two smaller ones. More generally, consistency suggests that our level of discretization should not directly affect the probability that a laser cast of given a length will be interrupted.

If we define, $P_c(x, \rho)$ to be the cumulative probability that the laser will have been interrupted after traveling distance x through a medium of type ρ, our consistency condition can be stated more generally in terms of k divisions as,

$$P_c(x, \rho) = \sum_{i=1}^{k} P_c(\frac{x}{k}, \rho)(1 - P_c(\frac{x}{k}, \rho))^{(i-1)}$$

This summation represents that the laser could be interrupted during any segment of length $\frac{x}{k}$. We accumulate the probability of stopping in each one. Inside the summation, the first term is the probability that the scan would be obstructed in the given segment. The second term represents the probability of the scan making it that far; that is, the probability that each previous segment did *not* obstruct the laser. As can be seen by simple substitution, the exponential distribution, $P_c(x, \rho) = 1 - e^{\frac{-x}{\rho}}$, for a positive scalar ρ, is a compact representation which satisfies this consistency condition. We will refer to ρ as the *opacity* of grid square.

For any laser cast, we can express the interaction between the model and the trace of the cast, up to some point n, as a vector of distances traveled through grid squares $\mathbf{x} = (x_1 \ldots x_n)$ and the corresponding opacities $\boldsymbol{\rho} = (\rho_1 \ldots \rho_n)$ of those squares. As noted earlier, the distances will not be uniform. Depending upon the angle of a cast, the beam could cut across a grid square in an axis parallel manner, go across it diagonally, or possibly just clip a corner of the square. The probability that a laser cast will be stopped at some point along its trajectory is therefore equal to the cumulative probability that the laser cast is interrupted by squares up to and including n:

$$P(stopped = true) = P_c(\mathbf{x}, \boldsymbol{\rho}) = \sum_{i=1}^{n} P_c(x_i, \rho_i) \prod_{j=1}^{i-1} (1 - P_c(x_j, \rho_j)).$$

We express the probability that the laser cast will be interrupted at grid square j as $P(\text{stop} = j)$, which is computed as the probability that the laser has reached square $j-1$ and then stopped at j,

$$P(\text{stop} = j | \mathbf{x}, \boldsymbol{\rho}) = P_c(x_j, \rho_j)(1 - P_c(\mathbf{x}_{1:j-1}, \boldsymbol{\rho}_{1:j-1})),$$

where $\mathbf{x}_{1:j-1}$ and $\boldsymbol{\rho}_{1:j-1}$ have the natural interpretation as fragments of the $\mathbf{x}$ and $\boldsymbol{\rho}$ vectors.

Suppose the vector $\boldsymbol{\delta} = (\delta_1 \ldots \delta_n)$ is a vector of differences such that δ_i is the distance between the laser distance measurement and grid square i along the trace of the laser cast. We express the conditional probability of the measurement, given that the laser beam that was interrupted in square i, as $P_{\mathcal{L}}(\delta_i | \text{stop} = i)$, for which we make the typical assumption of normally distributed measurement noise [25]. Note the the δ_i terms are only defined if the laser measurement observes a specific stopping point.

The events $(\delta_i, stop = i)$ for all i form a partition of all possible laser stopping events. Therefore, the probability of the laser measurement, L, with an observed stopping point, is then the sum, over all grid squares in the range of the laser, of the product of the conditional probability of the measurement given that the beam has stopped at that point, and the

probability that the beam stopped in each square,

$$P(L, \text{stopped} = \text{true}) = \sum_{i=1}^{n} P(\delta_i, \text{stop} = i | \mathbf{x}, \boldsymbol{\rho}) = \sum_{i=1}^{n} P_{\mathcal{L}}(\delta_i | \text{stop} = i) P(\text{stop} = i | \mathbf{x}, \boldsymbol{\rho}).$$

3.5 Map Updates & Observation Model

The mean of the exponential distribution with opacity ρ is simply ρ, which makes updating our map particularly simple. For each square, we maintain what is essentially a laser odometer that sums the total distance d_τ traveled by laser scans through the square. We also keep track of the number of times that the laser is presumed to have stopped in the square, s. Our estimate of ρ is therefore $\hat{\rho} = \frac{d_\tau}{s}$. In our initial implementation of the occupancy grid we treat the laser as a reliable measurement when updating the map. Thus, stop counts and grid odometers are updated under the assumption that the reported laser depth is correct. We realize that this remains somewhat contradictory to our observation model and we plan to implement soft updates, which will bring our approach closer to an incremental version of EM, in future work.

Grid squares which have not been observed before are treated in a special manner. The difficulty with such squares is that it is difficult to estimate the probability of the laser stopping in a given square without previous experience with that square. The usual solution to such a problem is the use of a prior. For our purposes, we use a gamma distribution as a conjugate prior, with a shape parameter of 1, and a scale parameter that has a natural interpretation as a previously observed ratio of distances to stops. We would like to set this ratio to one stop every 8m (the maximum range of our laser range finder. The strength of this prior is also kept low, equivalent to observing this probability for a few centimeters worth of distance. This gives us a prior of $d_{prior} = 0.04$m and $s_{prior} = 0.005$.

Since all grid squares are at least semi-transparent, the line trace will never terminate on its own; there is always a some chance that the laser should not have been stopped yet.

However, we know that when the laser reports an obstruction, there must be an object at some point within its range. Therefore, when a line trace reaches the maximum range of the sensor, we normalize the probability of the observation by the total probability that the laser has been stopped so far. This has the effect of enforcing the assumption that a finite laser reading must have originated from a physical object somewhere along its trajectory:

$$P(L|\text{stopped} = \text{true}) = \frac{P(L, \text{stopped} = \text{true})}{P(stopped = true)} = \frac{P(L, \text{stopped} = \text{true})}{P_c(\mathbf{x}, \boldsymbol{\rho}\,)}.$$

Conversely, when the laser is not observed to be stopped, the probability of the laser scan is equal to the probability that no object along its trajectory obstructed the laser:

$$P(L|stopped = \text{false}) = 1 - P_c(\mathbf{x}, \boldsymbol{\rho}\,).$$

This proposed observation model assumes that all of the errors in the observations are normally distributed. Since each observation is independent, a single poor observation can have a drastic impact on the total probability of a given pose. However, in reality, there exist many other causes for disagreements between the observations and the map. Reflections from specular surfaces, non-static objects, and discretization errors are but a few sources which can give rise to highly erratic readings. Fortunately, these are all fairly low probability events, and do not need to be handled specifically by the observation model. However, their impact on the total probability should be limited. Therefore, a certain amount of "background noise", ϵ, is allowed in the observations. Any single observation has a lower limit imposed on its probability, which can be interpreted as the probability of an unmodeled event causing an erratic reading. In practice, this background noise level was set to be relatively low compared to the normal observation model, less than 0.5%:

$$P_{final}(L) = max(\epsilon, P(L))$$

Chapter 4

DP-SLAM

4.1 Algorithm

DP-SLAM implements a particle filter over maps and robot poses, using an occupancy grid to represent the map, to track the placement of objects in the environment. In this sense it is a direct extension of many successful localization algorithms [9, 2, 25, 18]. However, it is important to note that for pure localization, each particle is tracking just the robot's pose. Each particle exists in the same map, and therefore where a particle was resampled from in the last time step is irrelevant. How a particle arrives at its current location in the map is unimportant; all that matters is that the current pose is accurate.

When using a particle filter for SLAM, each particle corresponds to a specific trajectory through the environment and has a specific map associated with it. When a particle is resampled, the entire map itself is treated as part of the hidden state that is being tracked. Since the ancestry of each particle matters, different particles are not interchangeable in the same sense as in pure localization, and a particle filter for SLAM is less able to recover from mistakes. Small errors in the robot pose can be reinforced in later iterations, and can quickly accumulate to form a poor map. Therefore, significantly greater numbers of particles are necessary in order to cover the state space sufficiently, not only in order to cover the extra dimension being tracked across the space of maps, but also to ensure that at each iteration we are sufficiently close to the robot's "true" pose at that time, so as to not introduce a tendency towards error into the map.

For a particle filter to track this joint distribution properly over both robot poses and maps, it is necessary for each particle to maintain a separate, complete map. During the resampling phase of the particle filter, each particle could be resampled multiple times.

Therefore, it is necessary to copy over the entire map to each new particle, to allow each hypothesis to maintain its own set of updates to the map. This joint distribution, and the corresponding need for multiple map hypotheses, has been recognized for some time as being crucial to the SLAM process [26]. However, a direct approach to this method, where a complete map is assigned to each particle, is impractical. If the map is an occupancy grid of size M and P particles are maintained by the particle filter, then ignoring the cost of localization, $O(MP)$ operations must be performed during the resampling step merely copying maps. For a number of particles sufficient to achieve precise localization in a reasonably sized environment, this naive approach would require gigabytes worth of data movement per update.

The major contribution of DP-SLAM over previous attempts is an efficient representation of the map to make map copying more efficient, while at the same time reducing the overall memory required to represent these large numbers of occupancy grids. This is achieved through a method called distributed particle mapping (DP-Mapping), which exploits the significant redundancies between the different maps.

4.1.1 Single Map

To appreciate fully the need for multiple maps, we first consider the behavior of using a single map to track the observed environment. This is a common trick used by earlier researchers in an attempt to avoid the complexities of map copying. In this method, a single map is maintained and used for localization for all of the particles. At each iteration, the single most likely particle is chosen, and the map is updated only once, based upon this greedy choice of robot pose. All of the other particles are ignored during the mapping stage. This simple method is nice because it looks almost exactly like a problem of pure localization, and can be expected to take equivalent resources. The problem with it is that although the map may seem consistent for short lengths of motion, small errors can very

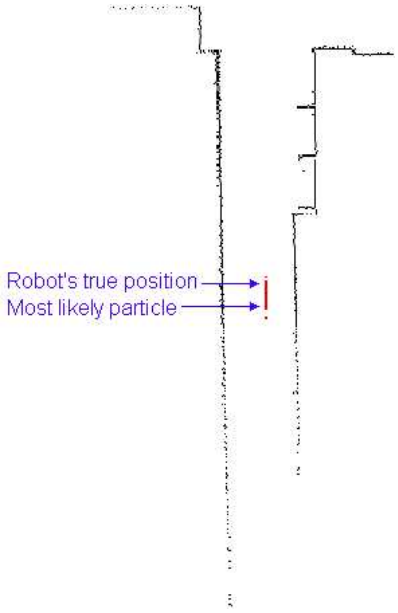

Figure 4.1: A SLAM algorithm in progress, demonstrating the distribution of particles. This illustrates the possible differences between the most likely particle at a given time step, and the true pose. This partial map corresponds to the upper left corner of the final map in Figure 4.2.

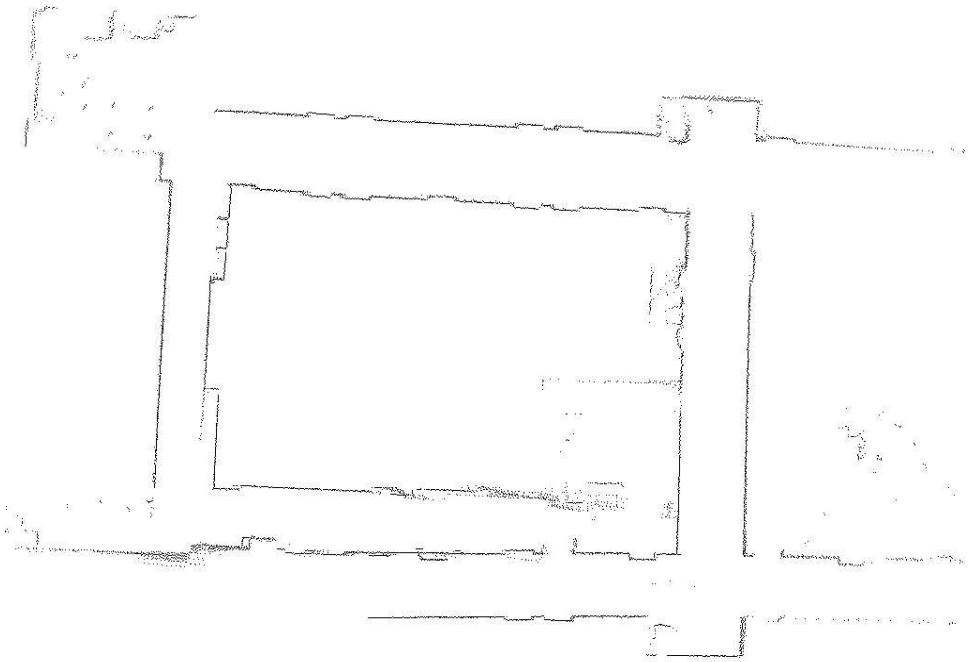

Figure 4.2: The results of ignoring the joint distribution over maps and robot poses, and maintaining only a single map. The two sections of hallway at the bottom are supposed to line up.

quickly accumulate, to give poor maps. The single map has no ability to revise the placement of earlier hypotheses as future information is acquired. It must make the assumption that that the most likely pose at each time step is the correct one, an assumption which is usually false. Any future observations are unable to influence this choice of the map update. Thus this method of mapping is very brittle and unable to recover from ambiguous situations.

Let us take a closer look at how this method performs in an actual SLAM situation. In Figure 4.1 we can see the results of a single-map SLAM algorithm in the midst of mapping a section of hallway. Here we see an overhead view of the world, with the edges of objects (occupied grid squares) depicted in black. In the center of the hall is a spread of red points, representing the current set of particles within the map. Since the hallway at this point is relatively featureless, it is difficult for the algorithm to differentiate its position along the length of the hall, and thus the particles are mostly distributed in a line along the length of the hallway. The most likely particle at this step, according to the sensor model, is a point near the top of the distribution. However, the robot's true position along the hallway was actually measured to be much closer to the bottom of the distribution. Therefore, updating the map based upon our single most likely particle has the effect of foreshortening the hallway slightly, and moving all future observations that much closer to the top of this hallway when they are entered into the map. Lacking any ability to keep multiple map hypotheses, future observations which could otherwise help resolve this ambiguity are still unable to correct the map.

In Figure 4.2, we can see the end result of this error, combined with many others like it. Here, the robot has completed a loop of the hallway, and has physically returned to its starting position in the world. However, because the SLAM algorithm was inaccurate, the robot's map does not reflect this. The two displaced sections of hallway at the bottom of this map are in fact different views of the same hallway, and should be aligned in an accu-

rate map. Unfortunately, since the single map SLAM method does not properly maintain the joint distribution over both robot poses and maps, it is unable to complete this task.

4.1.2 Ancestry Trees

It should be clear to the reader by now that correctly maintaining this joint distribution over maps and poses, and thus maintaining multiple map hypotheses, is crucial to solving the SLAM problem accurately, and producing high quality maps. However, as we have already noted, if implemented in a naive fashion, this can be a very expensive task, both in terms of processor time and memory requirements. The bottleneck of these approaches is of course the sheer size of the maps, and the cost of copying over each map when resampling particles.

An astute reader has most likely observed that the naive approach is doing too much work. Each map carries much in common with most of the other maps, and reproducing all of this information multiple times is an inefficient use of resources. To make this concept clearer, we will introduce the notion of a *particle ancestry*. When a particle is sampled from the distribution at iteration i to produce a new successor particle at iteration $i+1$, we call the generation i particle a *parent* and the generation $i+1$ particle a *child*. Two children with the same parent are *siblings*. From here, the concept of a particle ancestry extends naturally. To see how this can be useful, suppose that the laser sweeps out an area of size $A \ll M$ and consider two siblings, s_1 and s_2. Each sibling will correspond to a different robot pose and will make at most A updates to the map it inherits from its parent. Thus, the maps for s_1 and s_2 can differ in at most A map positions. The entire remainder of the map is identical.

When the problem is presented in this manner, the natural reaction from most computer scientists is to propose recording the "diff" between maps, i.e, recording a list of changes that each particle makes to its parent's map. While this would solve the problem of mak-

ing efficient map updates, it would create a bad computational problem for localization: Tracing a line though the map to look for an obstacle would require working through the current particle's entire ancestry and consulting the stored list of differences for each particle in the ancestry. The complexity of this operation would be linear in the number of iterations of the particle filter. The challenge is, therefore, to provide data structures that permit efficient updates to the map *and* efficient localization queries with time complexity that is independent of the number of iterations of the particle filter. We call our solution to this problem *Distributed Particle Mapping* or *DP-Mapping*, and we explain it in terms of the two data structures that are maintained: the ancestry tree and the map itself.

4.1.3 Maintaining the Particle Ancestry Tree

The basic idea of the particle ancestry tree is fairly straightforward. Every node in this tree represents a distinct hypothesis, possibly originating from a previous iteration of the particle filter. The tree itself is rooted with an initial particle, of which all other particles are progeny. Each particle maintains a pointer to its parent and is assigned a unique numerical ID. Finally, each particle maintains a list of grid squares that it has updated.

The details of how we will use the ancestry tree for localization are described in the subsequent section. In this section we focus on the maintenance of the ancestry tree, specifically on making certain that the tree has bounded size regardless of the number of iterations of the particle filter.

We maintain a bounded size tree by pruning away unnecessary nodes. First, note that certain particles may not have children, due to resampling, and can simply be removed from the tree. Of course, the removal of such a particle may leave its parent without children as well, and we can recursively prune away dead branches of the tree. After pruning, it is obvious that the only particles which are stored in our ancestry tree are exactly those particles which are ancestors of the current generation of particles.

This is still somewhat more information than we need to remember. What we truly are interested in is where and how the particles differ from each other in terms of the robot's trajectory, and thus the map. Therefore, if a particle has only one child in our ancestry tree, we can essentially remove it, by collapsing that branch of the tree. This has the effect of merging the parent's and child's updates to the map, a process described in the subsequent section. By applying this process to the entire tree after pruning, we obtain a *minimal* ancestry tree, which has several desirable and easily provable properties:

Independent of the number of iterations of the particle filter, a minimal ancestry tree of P particles

- has exactly P leaves, which are the set of particles for the current iteration;
- has branching factor of at least 2, as a direct consequence of our tree maintenance;
- consequently, it has depth no more than P.

The process of maintaining an ancestry tree is perhaps best illustrated through a simple example. Figure 4.3 depicts the start of this process. Here, the robot is traveling down a featureless hallway, and we will observe how the ancestry tree is updated as the robot moves. At the top of the figure is a single particle, where the robot's pose is represented by a red dot, and the current map additions are shown in black. This one particle is resampled from several times, to give a number of identical children. Since the map for each particle is inherited from their parent, it is shown in grey, instead of black. These new particles are then each propagated forward by means of the probabilistic motion model. Thus in Figure 4.4 each particle represents a different pose, and consequently, each has a different set of map updates. These particles are then scored, based on how well the new updates agree with the existing map, and randomly resampled proportionately based upon these weights (Figure 4.5).

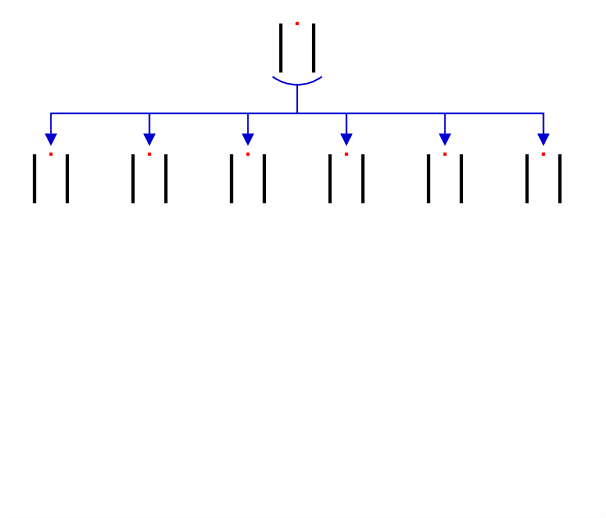

Figure 4.3: An ancestry tree just beginning

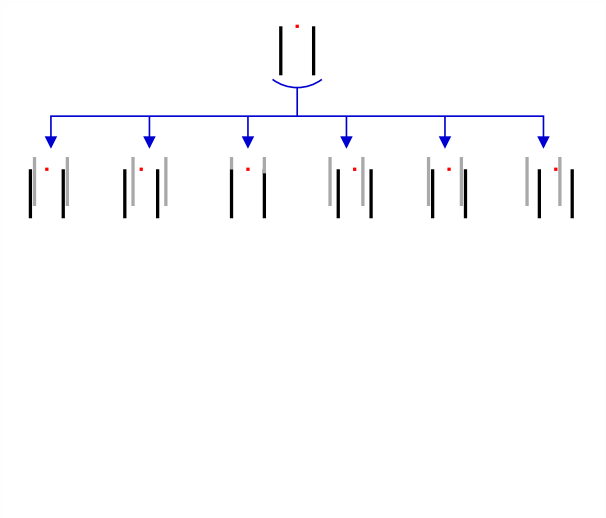

Figure 4.4: The children particles are propagated through the particle filter.

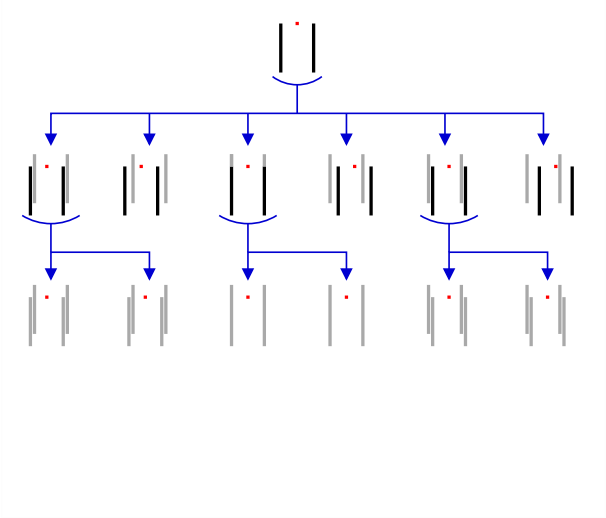

Figure 4.5: The child particles are resampled for the next generation

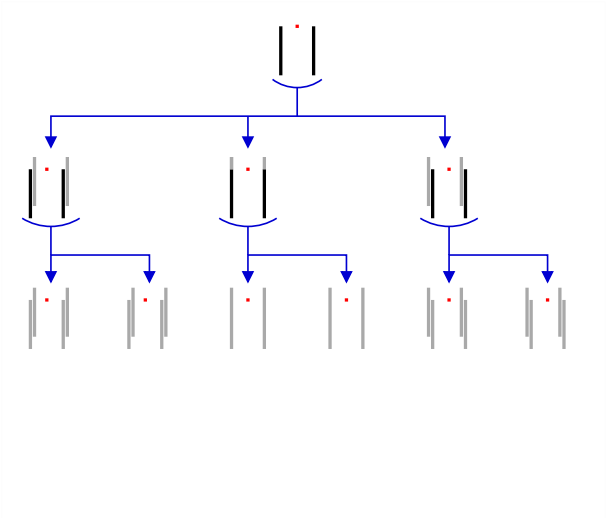

Figure 4.6: Unnecessary ancestor particles are pruned.

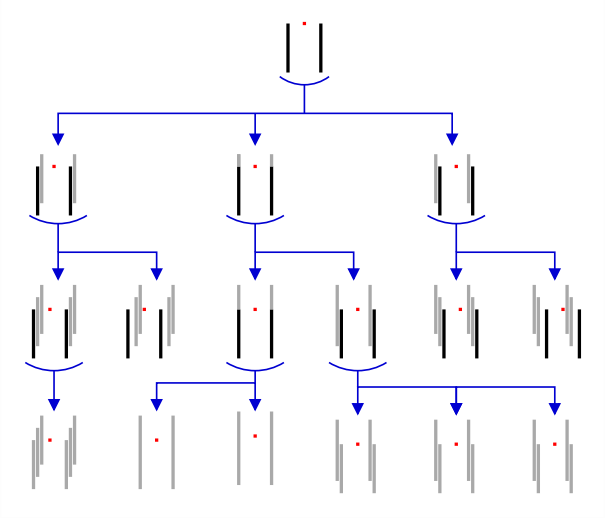

Figure 4.7: Resampling in the next generation

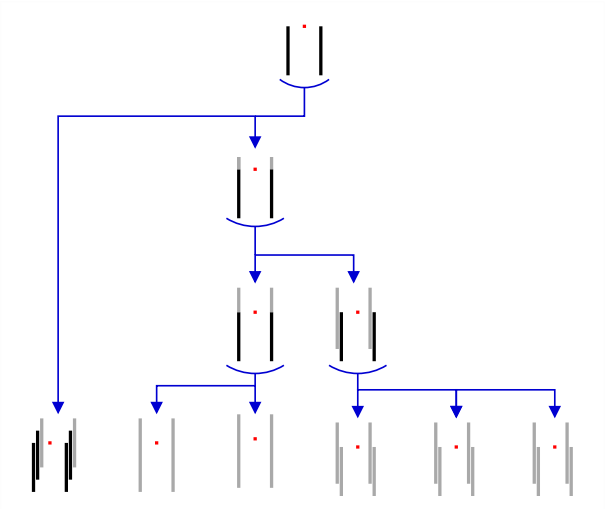

Figure 4.8: Irrelevant ancestors are pruned, and the column on the left is collapsed.

Notice that at this point some particles have scored higher than others, and therefore were resampled more than once. Since the total number of particles at each iteration is kept constant, this implies that there are other particles which were not resampled from at all. These childless particles can be removed from the ancestry tree, as they will have no influence on any future particles (Figure 4.6). In Figure 4.7, the new set of particles has again been propagated forward, and then scored and resampled, to give the next generation of particles. However, at this step, there are two important updates to be made to the ancestry tree. On the right, there are a pair of childless particles from the previous generation which can be removed from the ancestry tree. However, when this is done, their common parent will no longer have any children of its own. This older ancestor particle can also be removed from the tree, as it also now has no descendants in the current generation.

The other interesting update is on the left side of Figure 4.7. Here, we see that once the childless ancestor particles are removed, there will be a chain of ancestor particles, each of which will have only one child. Since we are interested only in how the current generation of particles diverge from each other, this chain gives us no pertinent information. Therefore, these nodes can all be merged into a single ancestor particle, effectively collapsing the chain. This single ancestor particle will contain all of the relevant map updates from all three ancestor particles, as well as representing the point at which its descendants diverged from the other particles. Figure 4.8 shows the pruned ancestry tree, with all unnecessary nodes removed, and the non-branching chains collapsed.

4.1.4 DP-Map Representation

The challenge for our map representation is to devise a data structure that permits efficient localization, updates and resampling. The naive approach of a complete map for each particle is inefficient due to the high cost of resampling, while the somewhat less naive approach of simply maintaining history of each particle's updates is also inefficient because

it introduces a dependency on the number of iterations of the particle filter during the localization stage.

Our solution to the map representation problem is to associate particles with maps, instead of associating maps with particles. DP-mapping maintains just a single occupancy grid, where the particles are distributed over the map. Unlike a traditional occupancy grid, which stores a single value at each grid square, we propose storing the entire set of observations for all of the particles. This set can initially be thought of being stored as a tree, keyed on the IDs of the particles that have made changes to the occupancy of the square. For efficient implementation, this tree can be kept balanced, using a red-black tree, or a similar method. In the next chapter, we will show a more efficient way of storing this information at each grid square.

The grid is initialized as a matrix of empty trees. When a particle makes an observation about a grid square it inserts its ID and the observation into the associated tree. Notice that this method of recording maps actually allows each particle to behave as if it has its own map. The only complication is that each lookup and update is now performed on a balanced tree, rather than a single entry. To check the value of a grid square, the particle checks each of its ancestors to find the most recent one that made an observation for that square. If no ancestor has made an entry, then the particle can treat this position as being unknown.

We can now describe the effects of collapsing an ancestor with a single child in the ancestry tree more precisely. The goal of this step is to merge all of the pertinent information from these two ancestry nodes, the parent and the child, into a single node. This allows for the removal of one of the nodes from the ancestry tree, preventing branches of the tree from growing arbitrarily, and thereby ensuring our $O(P)$ minimal bound on the size of the tree. We accomplish this through the following steps:

Each ancestry particle currently maintains a list of the grid squares it has observed,

and thus updated. For each item in the child's list:

  - Change the ID of the observation to match that of the parent.
  - If the parent also has an observation at this grid square, remove the observation associated with the parent.
  - Add this observation to the parent's list of observed squares.

- Redirect all children of the child node (i.e. grandchildren of the parent) to recognize the parent node as their new parent.
- Remove the child node from the ancestry tree.

4.1.5 SLAM using a DP-Map

Accessing a given grid square on a DP-map is slightly more complicated than with a standard occupancy grid. Since each grid square contains an entire set of observations, finding out whether a specific particle has made an observation at this location requires a search across this set of observations. However, each particle inherits the map updates inserted by its ancestors. Therefore, we need to step backwards through the particle's ancestry, comparing each ancestor to the set of observations at this grid square. This will give us the most recent map update for this location that this particle should have its own map.

Making an update to the map is a simpler process than accessing it. When a new observation is made, the updated value is added to the set of observations at that grid square. At the same time, the ancestor particle which made the update keeps a pointer to the specific observation in the map.

Deletions of entries to the map are only performed when an ancestor particle is removed from the ancestry tree. Using the list of map updates for this ancestor particle, which was created when the map updates were originally added to the map, we can easily remove all of the pertinent observations.

4.2 Complexity

The one nice thing about the naive approach of keeping a complete map for each particle is the simplicity: If we ignore the cost of copying maps, then lookups and changes to the map can all be done in constant time. In these areas, distributed particle mapping may initially seem less efficient, due to the search over observations. However, we can show that DP-maps are in fact asymptotically superior to the naive approach. We first present an analysis of the direct implementation of DP-SLAM, resulting in $O(AP^2logP)$ worst-case running time. With some careful changes, it is possible to improve this running time to $O(AP^2)$ [27]. However, that result is completely superseded by our results in the next chapter, where we show how the algorithm can be implemented much more efficiently. Those methods reduce the running time for DP-SLAM to $O(AP)$, arguably the most efficient bound that we could hope for.

4.2.1 Naive Implementation

As we have already noted, maintaining the joint distribution over maps and poses has been attempted before in SLAM. The naive approach to maintaining these multiple maps involved keeping a single full map for each particle. This has the nice feature of fast lookup time; accessing any given grid square for a particular particle can be done in constant time. If the area observed at each time step is bounded by A, and P particles are used, this means that the localization step of SLAM can be performed in $O(AP)$ time. Similarly, updating a grid square takes only constant time, and so the mapping step should only take $O(AP)$ time.

However, the main bulk of computation lies in the resampling step. Keeping multiple complete maps means that the robot will need to copy over an entire map for each particle, which will require $O(MP)$ time, where M is the size of the map. Since typically $M \gg A$, this obviously is the dominant term in the computation. Likewise, the memory

requirements for this method will be $O(MP)$ space. It is the enormous size of MP that quickly hampers the robot in implementation. Vast memory requirements and processing time needed for implementation have made this method infeasible for use in any reasonably sized environment.

4.2.2 Initial Analysis of DP-SLAM

The main goals for using distributed particle maps are to remove the costly step of copying P maps at every iteration, and simultaneously reduce the space requirements to a more reasonable bound. To achieve these goals, we are willing to accept an increase in the complexity of the other steps. As we will see later, this sacrifice is not in fact necessary.

Lookup on a DP-map requires a comparison between the ancestry of a particle with the observation tree at that grid square. Let D be the depth of the ancestry tree, and thus the maximum length of a particle's ancestry. Therefore, we can complete our lookup after just D accesses to the observation tree. Strictly speaking, as the ancestry tree is not guaranteed to be balanced, D can be as much as $O(P)$. However, in practice, this is almost never the case, and we have found $D \approx O(\log P)$, as the nature of particle resampling lends to very balanced ancestry trees. Since the observation tree itself can hold at most P entries, and a single search takes $O(\log P)$ time. Accessing a specific grid square in the map can therefore be done in $O(D \log P)$ time.

For localization, each particle will need to make $O(A)$ accesses to the map. As each particle needs to access the entire observed space for its own map, we need $O(AP)$ accesses, giving localization with DP-maps a complexity of $O(ADP \log P)$.

To complete the analysis we must handle two remaining details: The cost of inserting new information into the map, and the cost of maintaining the ancestry tree. Since the observation tree for each grid square is balanced, insertions and deletions on our map both take $O(\log P)$ per entry. Each particle can make at most $O(A)$ new entries, which in turn

will only need to be removed once. Each new map update requires accessing the map, in order to find the previous entry for this particle. Thus the procedure of adding new entries can be accomplished in $O(AD \log P)$ per particle, or $O(ADP \log P)$ total, and the cost of deleting childless particles will be amortized as $O(ADP \log P)$.

It remains to be shown that the housekeeping required to maintain the ancestry tree has reasonable cost. Specifically, we need to show that the cost of collapsing childless ancestry tree nodes does not exceed $O(ADP \log P)$. This may not be obvious at first, since successive collapsing operations can make the set of updated squares for a node in the ancestry tree as large as the entire map. We now argue that the amortized cost of these changes will be $O(ADP \log P)$. First, consider the cost of merging the child's list of modified squares into the parent's list. If the child has modified n squares, we must perform $O(n \log P)$ operations (n observation tree queries on the parent's key) to check the child's entries against the parent's for duplicates.

The final step that is required consists of updating the ID for all of the child's map entries. This is accomplished by deleting the observation with the old ID, and inserting a new copy of it with the parent's ID. The cost of this is again $O(n \log P)$. Consider that each map entry stored in the particle ancestry tree has a total potential of D times that it can be involved in a collapse over the course of the algorithm, since D is the total number of nodes between its initial position and the root, and no new nodes will ever be added in between. At each iteration, P particles each create A new map entries with potential D. Thus the total potential at each iteration is $O(ADP \log P)$.

The computational complexity of DP-SLAM can be summarized as follows: For a particle filter that maintains P particles, with a laser that sweeps out A grid squares, and an ancestry tree of depth D, DP-SLAM requires:

- $O(ADP \log P)$ operations for localization arising from:
  - P particles checking A grid squares

  - A lookup cost of $O(D \log P)$ per grid square

- $O(ADP \log P)$ operations to insert new data into the tree, arising from:
  - P particles inserting information at A grid squares
  - Each insert requires a lookup to retrieve the previous information, at a cost of $O(D \log P)$
  - Insertion cost of $O(\log P)$ per new piece of information
- Ancestry tree maintenance with amortized cost $O(ADP \log P)$ arising from
  - A cost of $O(\log P)$ to remove an observation or move it up one level in the ancestry tree
  - A maximum potential of ADP introduced at each iteration.

The space complexity of DP-SLAM is a little harder to put a tight bound on. At first glance, it appears that the map could have a worst case scenario in which each grid square has a complete set of P different observations. Therefore, the space complexity of DP-SLAM could approach $O(MP)$, and be asymptotically no better than the naive implementation. However, the robot's uncertainty very rarely stretches far enough back in time to allow for such a case. Much more often, there is one recent iteration in the past, before which all of the current particles agree on the trajectory. This point is referred to as the *point of coalescence*, and will be described in more detail in Chapter 7. If we establish this point of common ancestry as existing C iterations in the past, then we can impose a very weak $O(M + ACP)$ memory bound on DP-SLAM, where $AC < M$. The important point to be made by this observation is that for larger maps, the memory requirements are closer to that of maintaining only a single map, instead of P separate ones.

4.2.3 Empirical Evaluation of DP-SLAM

To evaluate the results of DP-SLAM, we ran several experiments using three different methods. The first experiments were designed to compare the results from using a single-map SLAM algorithm, such as the one described in section 4.1.1, against the results of DP-SLAM, using a simple deterministic occupancy grid for both methods. The second set of experiments demonstrate the benefits of using the stochastic occupancy grid described earlier, over the deterministic occupancy grids.

These test were performed on data from sensor logs, in order to ensure consistency between experiments. The sensor logs were collected by our iRobot ATRV Jr. in a cyclic hallway environment, with observations made approximately every 15cm. The robot is equipped with a SICK laser range finder, which scans at a height of 7cm from the floor. Readings are made across 180°, spaced one degree apart, with an effective distance of up to 8m. The error in distance readings is typically less than 5mm. The logs were then run offline on a fast PC (2.4 GHz Pentium 4), in a manner simulating the actual run, in order to allow for better control in our experiments.

We examined results from two different environments. The first is a smaller loop of a carpeted office environment, from the D-Wing of Duke University's LSRC building, approximately 30m x 25m, with the robot traversing a path 16m x 14m. The second, larger loop is also from the LSRC building, including C-Wing and a catwalk above the foyer. This second environment is much more complicated, and includes both carpet and tiled floor, as well as transparent windows and many small, cluttered objects. The size of this environment is roughly 25m x 60m, with the robot completing a loop approximately 14m x 40m.

The sensor logs for these experiments, as well as additional experiments, annotated maps, and pictures are available at `http://www.cs.duke.edu/~parr/dpslam/`.

For the results we present, it is important to emphasize that our algorithm knows *abso-*

lutely nothing about the environment or the existence of loops. No assumptions about the environment are made, and no attempt is made to smooth over errors in the map when loops are closed. The precision in our maps results directly from the robustness of maintaining multiple maps.

Deterministic Maps

In our initial implementation our map representation is a deterministic occupancy grid. At first, we are ignoring the stochastic map representation described at length earlier in this document, and instead are allowing only two different occupancy values for the grid squares, either **occupied**, indicated in black, or **empty/unknown**, in white. In the next section, we will see the effects of a stochastic map representation.

This first sensor log that we use as a test of DP-SLAM is the smaller, less complicated environment. This is the same sensor log that we have used before, in order to demonstrate the failings of using only a single map for all particles (Figure 4.2 and repeated in Figure 4.9). In this test, the robot began in the middle of the bottom hallway and continued counterclockwise through the rest of the map, returning to the starting point. The result shown in Figure 4.10 shows the highest probability map generated by DP-SLAM after the completion of the D-Wing loop using 9000 particles.

This example demonstrates the accuracy of DP-SLAM when completing a loop, one of the more difficult tasks for SLAM algorithms. After traveling 60m, the robot is once again able to observe the same section of hallway in which it started. At that point, any accumulated error will readily become apparent, as it will lead to obvious misalignments in the corridor. As the figure shows, the loop was closed perfectly, with no discernible seam or misalignment.

To underscore the advantages of maintaining multiple maps, we refer the reader back to the results obtained when using a single map and the same number of particles. Figure 4.9

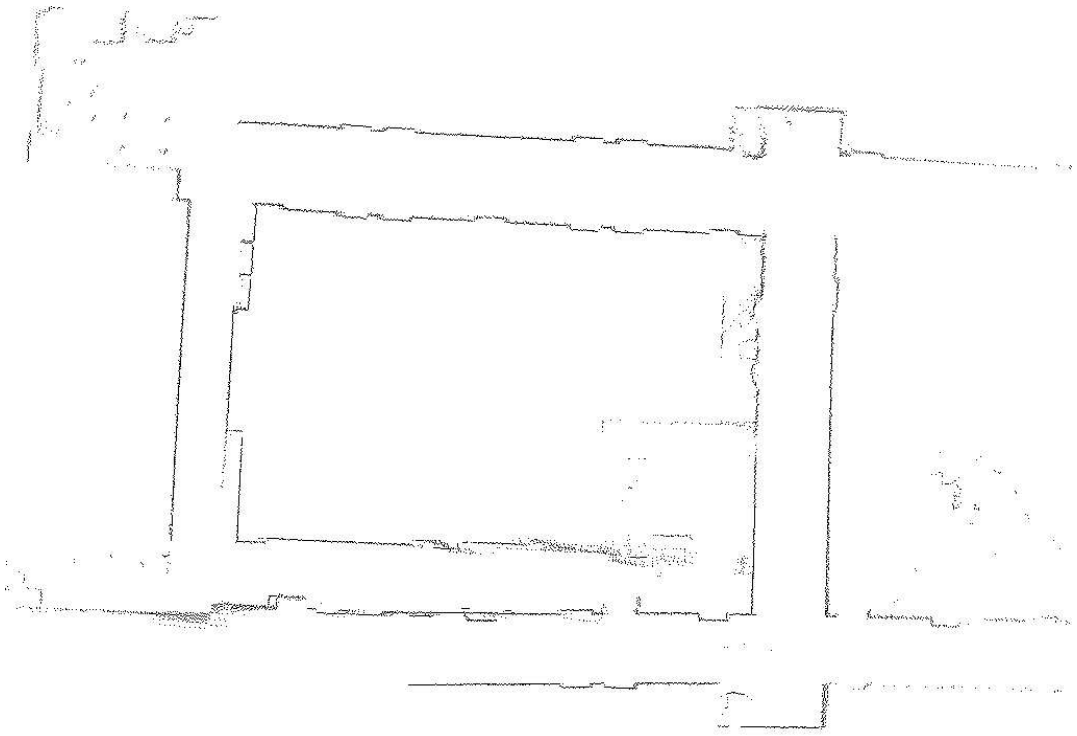

Figure 4.9: SLAM using a single map.

shows the result of processing the same sensor log file by generating 9000 particles at each time step, keeping the single particle with the highest posterior, and updating the map based upon the robot's pose in this particle. There is a considerable misalignment error where the loop is closed at the bottom of the map.

Stochastic Maps

Figure 4.10 demonstrates the successful performance of DP-SLAM using deterministic occupancy grids. However, as can be expected, deterministic maps have difficulties in larger, more complex situations. The previous D-Wing environment was fairly regular and well behaved – clear walls are always apparent, and most objects interact predictably with the laser range finder. In the C-Wing environment, walls are often cluttered and irregular, at times disappearing altogether, such as along the catwalk (see Figure 4.13). In addition, a number of windows are apparent, which are only semi-opaque to the laser, and the thin rails

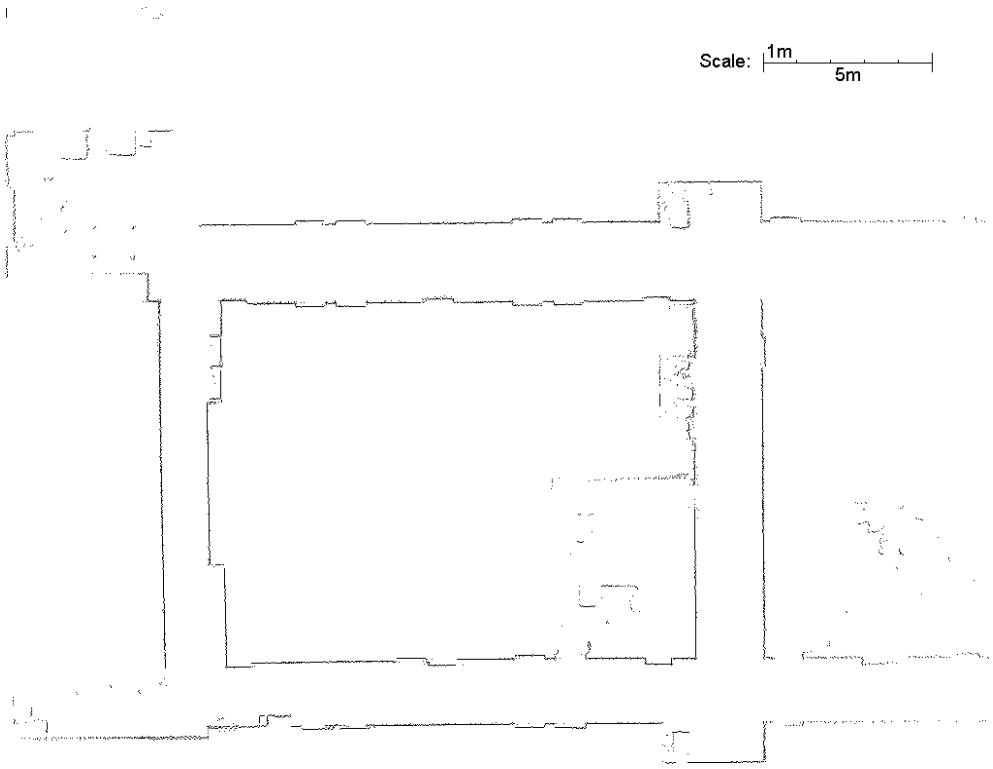

Figure 4.10: A DP-SLAM map with 9000 particles.

along the catwalk are smaller than the grid resolution for the map. This more complicated environment exposes the shortcomings of deterministic occupancy. Figure 4.11 shows what happens when this type of a map representation encounters the more challenging C-Wing section. The robot begins the run at the very top left corner of the map, and travels down the long hallway on the left, before turning up at the bottom of the map and returning along the right hallway. The loop is finally closed near the top right of the map. Predictably, the algorithm runs into serious trouble early into the run, which has continuing effects throughout the rest of the map.

When stochastic occupancy grids are used, a dramatic improvement is apparent. Figure 4.12 shows a map produced by stochastic occupancy grids with a resolution of 3cm per grid square, using 10,000 particles. We again emphasize that our algorithm knows *nothing* about loops and makes no explicit effort to correct map errors. The extraordinary precision and seamless nature of our maps arises solely from the robustness of maintaining a joint

Figure 4.11: Deterministic occupancy grids fail to handle the difficulties of C-Wing

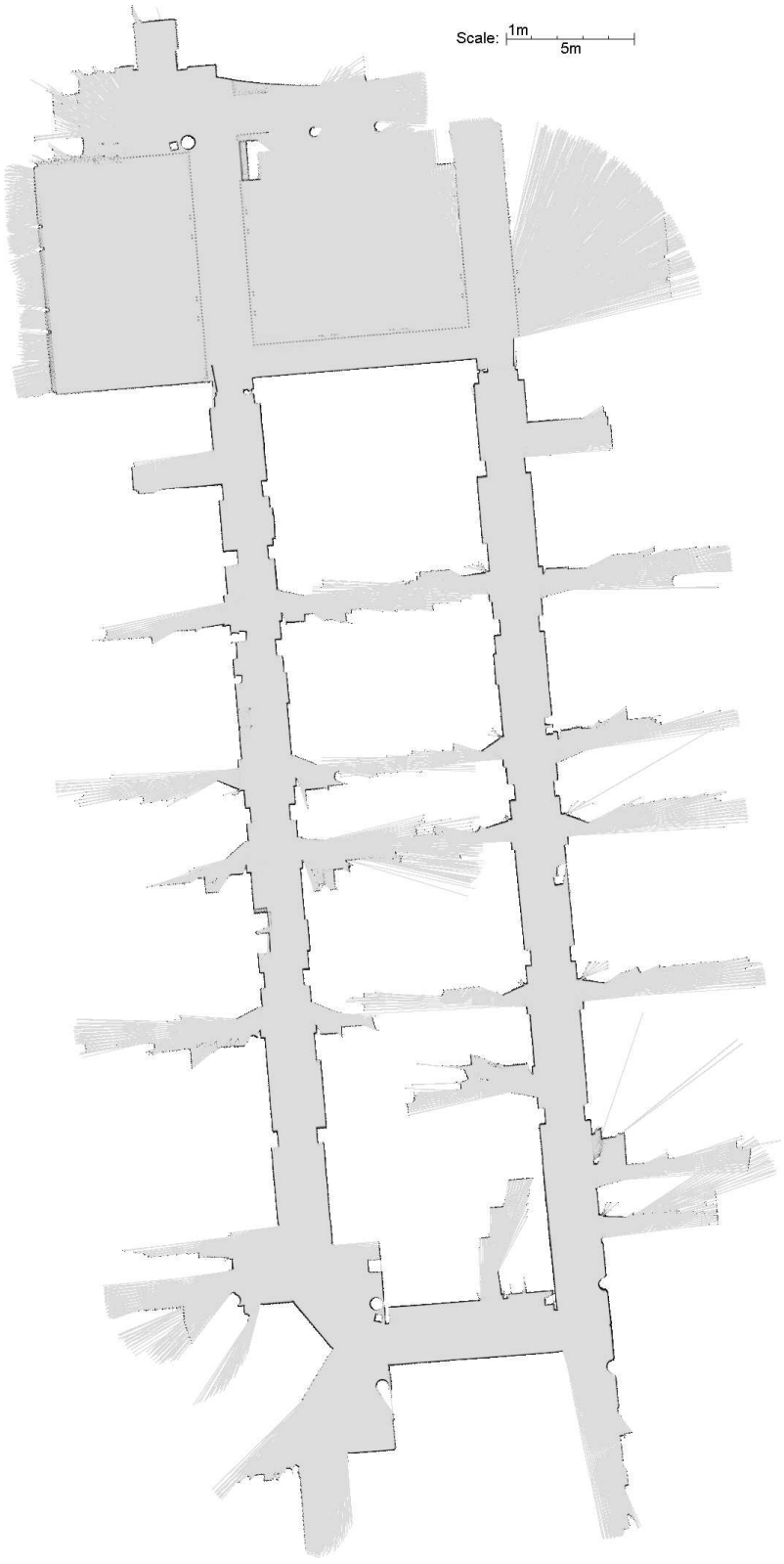

Figure 4.12: Proper stochastic mapping can successfully close the loop in C-Wing, using the same number of particles.

Figure 4.13: Robot perspective on the catwalk and railing, taken close to the *Railing* label in Figure 4.12. Slight changes in the robot position will affect which balusters are hit by the laser range finder, and which are missed.

distribution over robot positions and maps.

Details about the map, the environment, and the robot's trajectory give insight into the robustness and accuracy of the algorithm. The area in which the robot starts is on a raised catwalk, with a railing supported by many thin balusters (Figure 4.13), which at our scanning height appear as a series of small, evenly spaced obstacles. (The individual balusters are visible when the map is enlarged.) These small balusters provide a very difficult challenge for both localization and mapping, due to both their size and lack of distinction. Another challenge is a large set of windows along the left hand side of the map, near the top edge. The glass is a semi-opaque surface to the laser, only occasionally stopping the laser, due to dirt and angle of incidence. On close inspection, this area of the map may appear blurry, with some possible line doubling. This is actually because the window is double paned, and the laser has a chance of being stopped by either pane.

Other features of note are the intervening openings, which occur between the two long stretches of hall. When the robot moves from top to bottom along the left hallway, it sees only the lower walls of these passages, and it sees the upper walls on the return trip. Therefore, these passages provide no clues that would make it any easier for the robot to

close the large loop in the map.

The loop is closed on the top right of the map on a catwalk parallel to the first. The accuracy in this map is high enough to maintain the correct number of balusters for the hand rail between the two catwalks. There is one section of map that may appear to be inaccurate, at the bottom right hand corner of the map. Here, there are two intersecting hallways, which meet at a slightly acute angle. This is enough of a disparity to make one end of the two "parallel" hallways in our map approximately 20cm closer together on the top end when compared to the bottom. We were (pleasantly) surprised to discover, upon measuring the corresponding areas in the real world, that there was in fact a disparity of approximately 20cm in the real building. Our algorithm had detected an anomaly in the construction.

We performed several other experiments on the same data log. Our first test involved the same algorithm with a grid resolution of 5cm. With the same parameters, and an equal number of particles, the loop still successfully closed. Our last experiment involved a more simplistic approach to representing occupancy probabilities, which merely used a ratio of the number times the laser had stopped in the square compared to the number of times that it was observed. This method, while still able to handle some of the uncertainty in the environment, had significant skew and misalignment errors upon closing the loop, as seen in Figure 4.14.

Figure 4.15 demonstrates the results of running the stochastic mapping method on the sensor log from D-Wing. Since the deterministic map was able to correctly map the environment, it is not surprising to see that the stochastic method was also able to close the loop correctly. The important result is that these results were obtained with significantly fewer particles, only 3000, when compared to the deterministic method, which required 9000 for the same accuracy. So not only do we find that stochastic maps help us handle larger, more difficult environments, but they are also capable of handling the simpler environments with

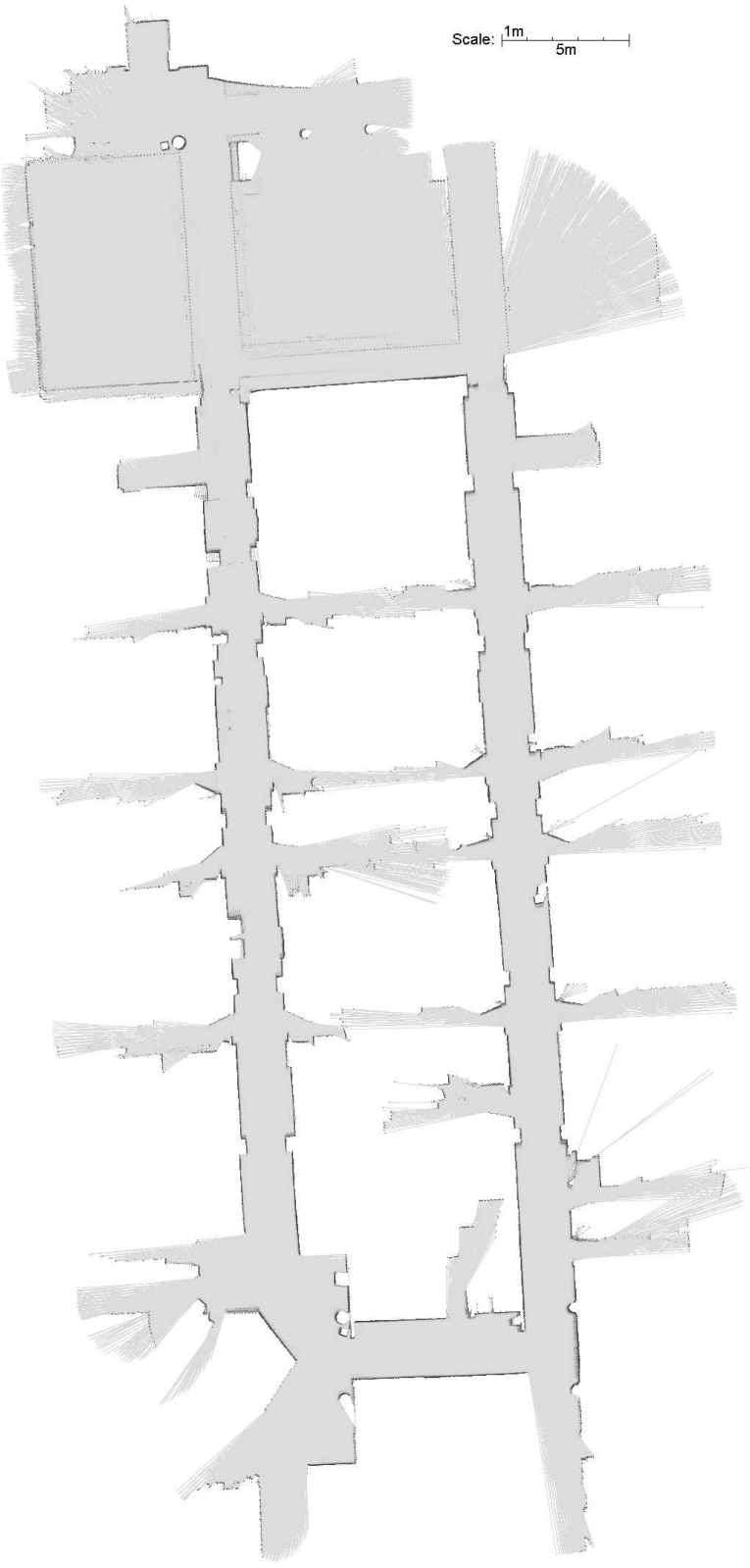

Figure 4.14: A simplistic occupancy grid of C-Wing.

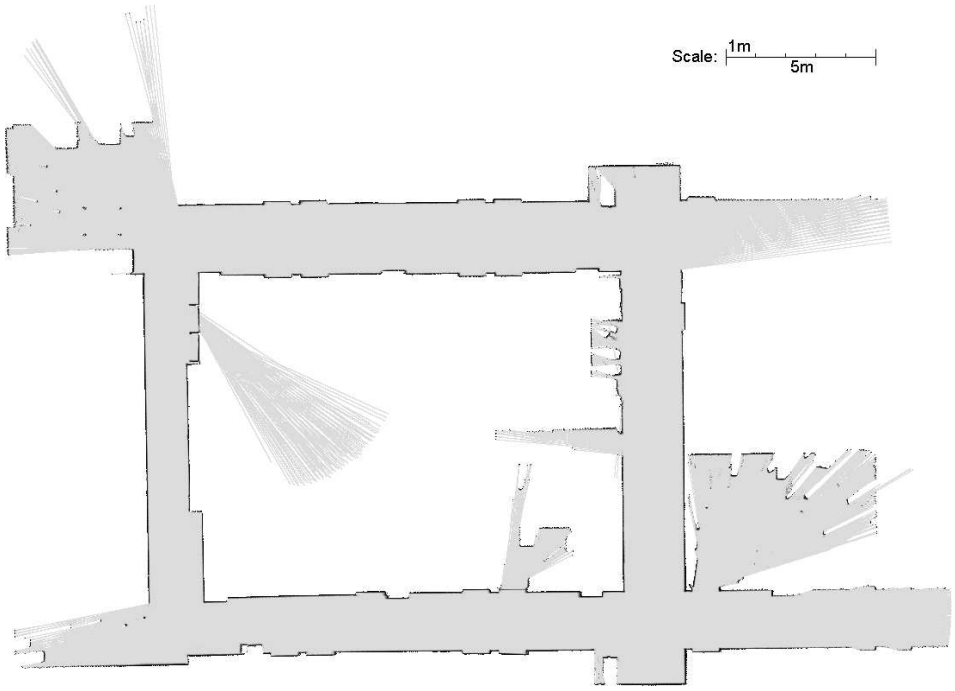

Figure 4.15: Stochastic occupancy grid of D-Wing, created using 3000 particles.

fewer particles. This demonstrates that it is not only those areas with semi-transparency and significant amounts of partial occupancies that benefit from the stochastic occupancy, but also objects with much better behaved surfaces as well.

Chapter 5

Linear Time Complexity

A straightforward implementation of the DP-SLAM algorithm can provide us with an asymptotic time complexity of $O(ADPlogP)$. As we have shown, this is good, but can we do better? To perform SLAM in larger, more complex environments, it becomes necessary to use more particles, and the feasibility for real-time implementation is threatened. A simple analysis suggests that $\Omega(AP)$ is a lower bound for any method that considers each grid square covered by A. How close can we get to this optimal bound, using the distributed particle map structure?

What we detail in this chapter is a clever manner of using dynamic programming in order to speed up computation, and in fact achieve this optimal $O(AP)$, under mild assumptions. In order to develop this efficient implementation, we first need to restructure how we store observations within the map. We can then construct an additional data structure, the *observation cache*, to allow for constant time references to the map for every particle.

5.1 Map Data Structure

Recall that the DP-SLAM map is a global occupancy grid-like array. Each grid cell holds a set of observations, with one entry for each ancestor particle that has made an observation of this grid square. Previously, this set of observations was stored as a balanced tree. However, this adds significant overhead, both conceptual and computational, to the algorithm and is not required in the linear time implementation of DP-SLAM. Instead, we simply store this information as a vector of observations, which we will refer to as the *observation list*. Each entry in this list is an *observation node* containing the following fields:

opacity A data structure storing sufficient statistics for the current estimate of the opacity of the grid cell to the laser range finder. For the purposes of the stochastic occupancy grids described earlier, this consists of the total observed laser distance traveled through this square, along with with the total number of times that the laser has been observed to stop in the square.

parent A pointer to another observation node in the same list, for which this node is an update. If this grid square had previously been unobserved by this particle's ancestry, this pointer is null. Note that if an ancestor of a current particle has seen this square already, then the opacity value for this square is considered an update to the previous value stored by the ancestor. However, both the update and the original observation can be stored, since it may not be the case that all successors of the ancestor have made updates to this square.

anode A pointer to the node in the ancestry tree which is associated with this observation.

In addition to this *observation list*, each grid also contains an **ocache pointer** and a **generation counter**, both used for the dynamic programming, and explained further later.

5.2 Ancestry tree node data structure

For reference purposes, we list the elements of the ancestry tree data structure. This is no different from the implementation described earlier, but is enumerated a little more specifically here. Each node in the ancestry tree consists of the following fields:

parent A pointer to this node's parent.

ID The unique identification number assigned to this ancestor particle.

children The number of nodes in the ancestry tree which are children of this node. When this number is reduced to zero, this node can be removed from the tree.

onodes A list of pointers to all of the observation nodes in the map which were added by this specific node.

5.3 Map cache data structure

The main sacrifice that was made when originally designing an occupancy grid for DP-SLAM was that map accesses could no longer be performed in constant time, due to the need to search across the set of observations at each given grid square. The map cache provides a way of returning to this constant time access, by reconstructing a separate local map for each particle. This local map is consistent with the history of map updates for that particle.

As Figure 5.1 indicates, the size of each of these local maps does not need to be very large. We only need to maintain in the cache those grid squares which are observable by the current sensor reading. Using a typical laser range finder that sweeps out a semicircular area, with a maximum laser range r, using a grid resolution of g, and assuming a reasonably concentrated posterior, the number of grid squares needed for any one local map in the cache would be approximately equal to $\frac{\pi r^2}{2g^2}$.

For a localization procedure using P particles and observing an area of A grid squares, there are a total of $O(AP)$ map accesses. For the constant time accesses provided by the map cache to be useful, the time complexity to build the map cache needs to be $O(AP)$. This result can be achieved by constructing the cache in two passes.

The first pass iterates over all grid squares in the global map which could be within the sensor range of the robot. For each one of these grid squares, the observation vector stores all of the observations made at that grid square by any particle. We traverse this vector, and for each observation visited, we update the corresponding local map with a pointer back to the corresponding observation node. This creates a set of partial local maps that have been seeded with their direct map updates, but no inherited map information. Since

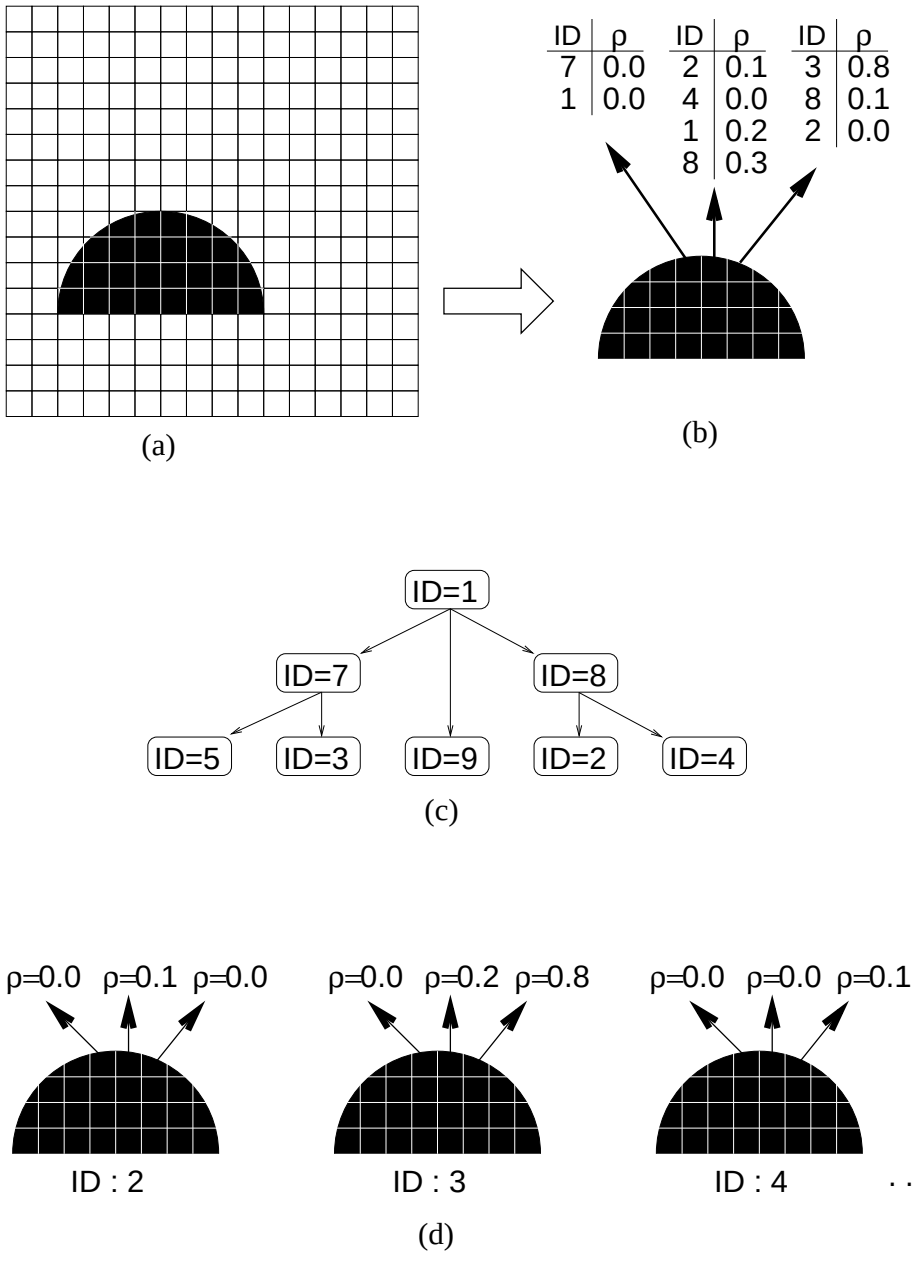

Figure 5.1: An illustration of how the map cache works. (a) The robot is only able to observe a small portion of the total occupancy grid. (b) Each grid square in the global map maintains an entire set of observations, identified by the ancestor particle which added that update. (c) The ancestry tree defines the how these observations are inherited. (d) The map cache maintains a complete local occupancy grid of the currently observed area for each leaf of this ancestry tree. Recall that the set of leaves in the ancestry tree defines the current set particles.

the size of the observation vector is no larger than the size of the ancestry tree, which has $O(P)$ nodes, the first pass takes $O(P)$ time per grid square.

In the second pass, we fill the holes in the local maps by propagating the inherited map information. The entire ancestry tree is traced, depth first, and the local map is checked for each ancestor node encountered. If the local map for the current ancestor node was not filled in during the first pass, then the hole is patched by the ancestor node's parent. Since we are tracing the ancestry tree from top to bottom, this will fill in any gaps in the local maps for grid squares that have been seen by any current particle. If a grid square has not been filled in by this process, then no ancestor of the given particle has observed this square, and it can be treated as unknown. As this pass is directly based on the size of the ancestry tree, it is also $O(P)$ per grid square. Therefore, the total complexity of building the map cache is $O(AP)$.

For each particle, this algorithm constructs a local map of pointers to the appropriate observation nodes. This provides constant time access to the opacity values consistent with each particle's map. Localization now becomes trivial with this representation. Laser scans are traced through the corresponding local map, and the necessary opacity values are extracted via the pointers. With the constant time accesses afforded by the local maps, the total localization cost in DP-SLAM is now $O(AP)$.

5.4 Updates

When the observations associated with a new particle's sensor sweep are integrated into the map, the following steps are performed for each grid square observed:

1. Retrieve the appropriate previous observation information for this grid square.

2. Insert a new observation node into the observation list for this grid square, which is an update of the information retrieved.

3. Insert a pointer to this new observation node into the particle's **onodes** list.

The cost of this operation is obviously no more than the cost of localization. In fact, the same map cache can be used to speed up the accesses for each grid square, thus saving a significant portion of the work.

5.5 Deletions

We will first consider the mechanism and cost of individual deletions from the ancestry tree, and then consider the computational impact of performing multiple deletions within a single iteration. There are two situations that require deleting nodes from the ancestry tree. The first is the simple case of removing a node from which the particle filter has not resampled. This involves the constant time operation of removing the ancestry node from the ancestry tree, and the process of removing the observations associated with this node from the observation lists in the global grid. Since pointers to these are stored at the node, this simply involves $O(A)$ constant time deletions from the observation list.

The second case for deleting a node occurs when an only child is merged with its parent in the ancestry tree. This involves the following steps:

1. Remove the child node from the ancestry tree.

2. For each observation node in the child's **onodes** list.

   (a) Replace the opacity value of the observation node's parent with opacity of the current child's observation node.

   (b) Remove the child's observation node from its grid cell observation list.

Note that merging has the effect of replacing the parent's map with the child's map. This is the desired behavior, since the child's map is more current than the parent's map

and there are no siblings left to inherent anything from the parent map. As in the case of a simple deletion, this involves $O(A)$ constant time deletions.

So far, we have discussed only the cost of individual deletions, but determining the overall time complexity of deletions is somewhat complicated because the number of deletions performed at any mapping iteration can vary quite significantly. When an ambiguity is resolved, entire branches of the ancestry tree could require deletion and/or merging. For this reason, we analyze the complexity of deletions in an amortized sense.

In the case of simple deletions, we establish a potential number of deletions based on the total information inserted per iteration. Notice that each iteration creates precisely P new ancestry nodes. Each added node has $O(A)$ new observations associated with it. Since each new observation can only be removed once, this introduces $O(AP)$ potential deletions at each iteration.

For the case of merges, remember that a merge only occurs when a parent has exactly one child. Therefore, only a single set of $O(A)$ observations from a given iteration can be inherited up an edge in the ancestry tree. This means that the total potential for merges introduced by any iteration of the particle filter is bounded by the number of interior nodes in the tree, which is $O(P)$. Since the contribution to each merge from a given iteration is $O(A)$, the total potential merging work introduced at each iteration is just $O(AP)$, and the amortized cost of all deletion-related work is just $O(AP)$ per iteration.

5.6 Summary of Computational Complexity

We have shown that localizing, adding new observations, and maintaining a minimal ancestry tree each require $O(AP)$ time. Ironically, the only part of the our algorithm with superlinear complexity is the simplest part of the resampling phase of the particle filter. For resampling, we must convert samples from the uniform distribution generated by our pseudo-random number generator to samples from our weighted set of particles. This is

log	particles	quadratic method	linear method
D-Wing	1500	55	14
D-Wing	3000	124	28
C-Wing	11000	1345	690
C-Wing	20000	3609	826

Table 5.1: Comparison of the running times for the original, quadratic version of DP-SLAM versus the linear implementation.

typically done by imposing a total ordering on the particles, computing the cumulative distribution, and then performing binary search to find the event corresponding to the sampled probability. For P particles, this would take $O(P \log P)$ time, so the overall complexity of our algorithm should be stated as $O(P \log P + AP)$. However, the area swept out by the laser will be much larger than $\log P$ for any typical laser and reasonable number of particles, so (AP) is accurate.

5.7 Implementation and Empirical Results

In the preceding section, we presented the algorithm in a manner intended to maximize clarity and ease of analysis. The only negative aspect of the observation cache is that it can be fairly memory intensive. Our actual implementation has the same computational complexity, but uses fewer pointers and fields to reduce memory consumption and constructs the observation cache for each grid cell in a lazy manner to improve speed.

For a complex algorithm like DP-SLAM, asymptotic analysis may not always give a complete picture of real world performance. Therefore, we provide a comparison of actual run times on our two different data logs, with several different particle counts.

The linear implementation is merely a more efficient method of arriving at the same numbers produced by a more straight forward implementation. Therefore, the maps produced by these two methods are identical, and displaying these maps would not be infor-

mative.

Chapter 6

Motion Models and Proposal Distributions

Robot motion models play an important role in modern robotic algorithms. The main goal of a motion model is to capture the relationship between a control input to the robot and a change in the robot's configuration. Good models will capture not only systematic errors, such as a repeated tendency of the robot to drift left or right when directed to move forward, but will also capture the stochastic nature of the motion. The same control inputs will almost never produce the same results and the effects of a robot's actions are, therefore, best described as distributions [9]. These distributions play an important role in algorithms that use particle filters for localization and mapping. Specifically, they form the proposal distribution for the particle filter.

In general, it is well known that a poor proposal distribution in a particle filter may require a prohibitively large number of particles to track the state of a system successfully. If the true behavior is not well within the sampling region of the particle filter, the probability of generating a particle consistent with the state of the system will be very low. In application, a SLAM procedure with a poor proposal distribution will require an excessive number of particles and yet may still lose track of the robot state. Thus, the motivation for acquiring a good motion model is quite strong.

6.1 Other Proposal Distribution Improvements

Poor proposal distributions are a common problem, and researchers have developed a number of methods for dealing with the issue. One common method used for particle filters in general is a technique called adaptive importance sampling [28]. This method takes a proposal distribution and evaluates a small number of samples within that distribution.

Those samples are then used to successively refine the proposal distribution, based upon their position and weight. This method has been effective in a number of other particle filter applications, but its usefulness for DP-SLAM is minimal. The improvement in the proposal distribution at each stage is not large enough to offset the cost of evaluation. This is exacerbated by the immense size of the joint distribution over maps and robot poses; each hypothetical map should have a slightly different proposal distribution after one step of refinement. In practice, it has been found that merely increasing the number of particles used is more efficient than attempting to refine the distribution using adaptive importance sampling.

Another method for improving the proposal distribution in SLAM relies on matching the observations from one time step to the next. Using scan matching to align the laser endpoints of one observation to the laser endpoints of the next, it is possible to find a better estimation of the motion than odometry alone will allow [29, 30]. Similar in concept to adaptive importance sampling, this method relies on the range sensor observations to refine the proposal distribution. Therefore, it has similar issues of a continued reliance on an initial distribution, and can require additional computational power. The major improvement for scan matching lies in the fact that it depends solely on the two most recent observations, and not on the map or the trajectory of the robot. Therefore, scan matching can be performed just once each iteration, and it will provide a refined distribution over possible poses for *a*ll particles. Adaptive importance sampling, on the other hand, is dependent on the map to give the refined distribution. Consequently, adaptive importance sampling must be performed for each particle separately, or else it will give a biased estimate, with greater variance.

6.2 Previous Calibration Methods

Previous work in automatic acquisition of motion models for mobile robots has been fairly sparse. Most of the efforts have dealt with the problem of systematic errors, rather than the levels of noise that can be present. Borenstein and Feng [31] describe a method for calibrating odometry to account for systematic errors. This method assumes a fairly smooth surface for calibration, with low non-systematic errors, and attempts to model each wheel independently. This method would become significantly more difficult for a robot with more than two drive wheels. Voyles and Khosla [32] use shape from motion to learn the motion model parameters, but instead of using the shaft encoders, attempt to model the observation of applied force vectors directly. This would require an additional sensor which is not typically available on many robots, and has limited accuracy. Roy and Thrun [33] propose a method which is more amenable to the problems of localization and SLAM. They treat the systematic errors in turning and movement as independent, and compute these errors for each time step by comparing the odometric readings with the position estimate given by a localization method. They can then use an exponential estimator to learn these two parameters online, assuming that short term localization results will be accurate enough to refine the motion model.

The goals of our approach are most similar to those of Roy and Thrun. We aim to have a method that can start with a crude model and bootstrap itself towards a more refined motion model, giving the robot the ability to adapt to changing motion parameters. Instead of merely learning two simple parameters for the motion model, as with the method proposed by Roy and Thrun, we seek to use a more general model which incorporates all of the interdependence between motion terms, including the influence of turns on lateral movement, and vice-versa. Furthermore, the proposed method extends the scope of the calibration beyond the systematic errors dealt with in previous methods. We believe that great gains in performance can be achieved by estimating the non-systematic errors, to quantify

the variance in the different movement terms. This can be crucial to the motion model of SLAM methods, as different amounts of noise in the movement terms can produce vastly different proposal distributions [34]. A properly calibrated set of variance parameters will provide the localization algorithm with a more appropriate proposal distribution, allowing it to better focus its resources on the most likely poses for the robot.

The algorithm for learning the motion model is integrated with a SLAM algorithm, giving increased autonomy to the system. The robot now has the potential to learn the most appropriate model based upon recent experiences, and in direct conjunction with its current task. This is especially useful as the robot's motion model will change over time, both from changes in the terrain and from general wear on the robot. It is also important that this calibration method can be performed in a remote location, without the need of external sensors to measure the robot's true motion. (A rover landing on another planet with unknown surface conditions would be an obvious application of this approach.)

With this view in mind, we can identify two categories of hidden variables in our problem formulation. We are attempting to learn both the map of the environment and the set of motion model parameters that describe stochastic relationship between the odometry and the actual movement of the robot. To estimate the parameters of this model, we propose using an EM algorithm: The expectation step is provided by a SLAM algorithm, implemented with some initial motion model parameters. The possible trajectories postulated are then used in the maximization step to create a set of parameters which best describe the motions represented by these trajectories.

6.3 Motion Model Details

Let the robot's pose at any given time step be represented as $\phi = (x, y, \theta)$, where θ is the facing angle of the robot. The motion model then seeks to determine $P(\phi'|\phi, o)$, where ϕ' is the robot's pose one time step in the future, and $o = (d, t)$ is the amount of lateral and

rotational movement (respectively) that odometry has reported over that time interval.

Roy and Thrun [33] propose the following motion model:

$$\begin{aligned} x' &= x + D\cos(\theta + T) \\ y' &= y + D\sin(\theta + T) \\ \theta' &= \theta + T \mod 2\pi \end{aligned}$$

Here, D is the actual distance traveled by the robot, and T is the actual turn performed. This is correct only if the turn and drive commands are performed independently, a simplifying assumption which even their own experiments violate. A simple improvement to account for simultaneous turning and lateral movement would be:

$$\begin{aligned} x' &= x + D\cos(\theta + (T/2)) \\ y' &= y + D\sin(\theta + (T/2)) \\ \theta' &= \theta + T \mod 2\pi \end{aligned}$$

This model assumes that the turning velocity of the robot is constant throughout the time step, and that the robot can only move in the direction it is facing. These improved equations do not take into account that, even in this case, the distance traveled will actually be an arc, and not a straight line. However, when T is reasonably small, this error is minor and can be absorbed as part of the noise.

A better model would take into account the ability of the robot to move in a direction that is not solely determined by the beginning and end facing angle of the robot. Such a model would be able to account for variable speed turns and sideways shifts, both of which have been apparent with our robots, even on the best of surfaces.

$$\begin{aligned} x' &= x + D\cos(\theta^*) \\ y' &= y + D\sin(\theta^*) \\ \theta' &= \theta + T \mod 2\pi, \end{aligned}$$

Here θ^* is the true movement angle of the robot. In this method, the direction of movement has been expressed separately from θ and T, which permits movement in a direction distinct from the facing angle of the robot. In practice it is often difficult to determine this independently from θ and T, but with some robots, the shaft encoders on each wheel can be read independently, and can give a more direct observation of this parameter.

Even in the rare cases where it might be possible to observe θ^*, it would be very difficult to develop a good noise model. Representing the noise in θ^* as a Gaussian would require some choice for a mean. For a robot which can perform holonomic turns, the lateral shift of the robot could very easily be in any direction, while the lateral movement reported by the odometry would be negligible. In this case, θ^* would more accurately be modeled as a uniform distribution. For these reasons, we prefer a slightly different model that decomposes the movement into two principle components:

$$\begin{aligned} x' &= x + D\cos(\theta + \frac{T}{2}) + C\cos(\theta + \frac{T+\pi}{2}) \\ y' &= y + D\sin(\theta + \frac{T}{2}) + C\sin(\theta + \frac{T+\pi}{2}) \\ \theta' &= \theta + T \mod 2\pi \end{aligned}$$

We approximate θ^* with $(\theta + \frac{T}{2})$ and refer to this direction as the *major axis* of movement. C is an extra lateral translation term, which is present to model shift in the orthogonal direction to the major axis, which we call the *minor axis*. This axis is at angle $(\theta + \frac{T+\pi}{2})$, and is defined so as to have a consistent (left-hand) orientation.

This motion model lends itself to a fairly natural noise model. We expect that the true values of D and T will be distributed normally with respect to the reported values, d and t, but that the mean of each will scale linearly with both d and t while the variance will scale with d^2 and t^2. This is plausible if the total noise is the sum of two independent noise sources with magnitude that scales linearly with d and t. We expect that C will have a similar dependence on d and t. In this view, C, D and T are all conditionally Gaussian

given d and t:

$$\begin{aligned} C &\sim \mathcal{N}(d\mu_{C_d} + t\mu_{C_t}, d^2\sigma^2_{C_d} + t^2\sigma^2_{C_t}) \\ D &\sim \mathcal{N}(d\mu_{D_d} + t\mu_{D_t}, d^2\sigma^2_{D_d} + t^2\sigma^2_{D_t}) \\ T &\sim \mathcal{N}(d\mu_{T_d} + t\mu_{T_t}, d^2\sigma^2_{T_d} + t^2\sigma^2_{T_t}), \end{aligned}$$

where μ_{A_b} is the coefficient for the contribution of odometry term b to the mean of the distribution over A. It is these sets of mean and variance terms that we propose to learn.

6.4 Parameter Estimation

The learning problem for our robot is that of discovering the parameters of the distribution $P(\phi'|\phi, w, o, m)$, where w is the reported odometry, o is the set of observations of the environment, and m is the map of the environment. With this in mind, consider a SLAM algorithm, which uses a particle filter to produce a distribution over maps and poses at each time step. For a given set of motion model parameters, our particle filter provides a set of possible trajectories with forward probabilities (normalized particle weights) at each time step. To complete the E step, we must perform backward smoothing over our particles. There are many ways to do this with a particle filter, but we use the simplest, which is to compute the probability of each trajectory and average across successor trajectories for particles that are resampled multiple times.

The complete run of the particle filter with smoothing can now be viewed as producing a set of N weighted estimates of the true motion of the robot, where sample i with weight w_i can be expressed as ΔX_i, ΔY_i, $\Delta\theta_i$. From algebraic manipulation of the previous

equations, we obtain

$$C_i = \frac{\Delta y_i - \Delta x_i \cdot tan(\theta_i)}{sin(\theta + \frac{\pi}{2}) - cos(\theta + \frac{\pi}{2})tan(\theta)} \quad (6.1)$$

$$D_i = \frac{\Delta x_i - C_i \cdot cos(\theta + \frac{\pi}{2})}{cos(\theta)} \quad (6.2)$$

$$T_i = \Delta\theta_i \quad (6.3)$$

To complete the M step of our EM procedure, we must compute the maximum likelihood values of the parameters in our model. The means in our model have linear contributions from the reported odometry values. We therefore determine the influence of each term on the motion parameters using a weighted least squares method. For example, let $\mu_{\mathbf{C}}$ be the column vector $[\mu_{C_d}\ \mu_{C_t}]^T$, $\mathbf{O}$ be an $N \times 2$ matrix, where each row is the reported odometric movement $[d_i\ t_i]$, and $\mathbf{C}$ be the $N \times 1$ matrix of the estimated C_i terms. We obtain the least squares solution of $\mu_{\mathbf{C}}$ from the overdetermined system:

$$\mathbf{WO}\mu_{\mathbf{C}} = \mathbf{WC},$$

where $\mathbf{W}$ is an $N \times N$ diagonal weight matrix with diagonal element i as $\sqrt{w_i}$. The process can be repeated for the other two motion terms.

The variance in our model has a quadratic dependence in the odometry terms. To compute the variance parameters for the C term in our model, $\sigma^{\mathbf{2}}_{\mathbf{C}} = [\sigma^2_{C_d}\ \sigma^2_{C_t}]^T$, we define $\mathbf{O^2}$ as an $N \times 2$ matrix whose rows are the squared odometry readings, $O_i = [d_i^2\ t_i^2]$. We define $\mathbf{C}^{\sigma^2}$ as the $N \times 1$ matrix such that $C_i^{\sigma^2} = (O_i\mu_{\mathbf{C}} - \mathbf{C_i})^2$. As before, we are interested in the least squares solution to an overdetermined system of linear equations:

$$\mathbf{WO^2}\sigma^{\mathbf{2}}_{\mathbf{C}} = \mathbf{WC}^{\sigma^2}.$$

The calculation is similar for the variance parameters of the other motion model terms.

The least squares solution for all 12 parameters of the motion model constitutes the M step of our EM procedure. The new model parameters can now be used for a new run

of the SLAM algorithm on the same set of sensor data, and the process can be repeated until (near) convergence. This method can be applied to varying quantities of motion data. Run over the entire set of data, it can be applied off-line as a means of determining the best motion model for a robot in future deployments in the same, or similar, environment. This is useful, as it allows the algorithm to learn, with high confidence, the proper set of motion parameters, due to the large amount of training data. It also provides the operator the ability to check the performance of final motion parameters, by observing the accuracy of the final trajectory.

An alternative, quasi real time application of this method would run EM on a smaller set of data, allowing the robot to learn the motion model as it explores. In this case, the robot would use a fixed size chunk of recent observations to fine tune the motion model to changes in its behavior. For example, the robot might apply this technique if it encounters a type of terrain that it has never seen before. This can slow any mapping activities undertaken by the robot. If the robot is using a SLAM algorithm for mapping, the EM nature of the model learning algorithm will require that the localization be run multiple times over each section and the mapping will no longer be real time. In practice, we expect that model tuning procedures would not be used continuously, but would be used primarily at sparse intervals or when there is some reason to believe that an inaccurate model is degrading mapping accuracy.

6.5 Empirical Results

We tested the learning algorithm on the D-Wing sensor log used in previous sections. During this set of experiments, we noticed a small anomaly where laser readings sometimes changed in a manner implying motion, when no changes were reported in odometry. This could possibly be caused by readings from the laser range finder not being perfectly synchronized with the readings from the from the odometers. Since the motion model de-

scribed is directly dependent on the magnitude of reported motion, the variance in these situations would be zero, and the SLAM algorithm would have no ability to recover the correct motion for that time step. To handle this problem, we found it necessary to set a minimum amount of noise that must be present at each time step. These levels were small (variances less than 2cm along the major axis and less than 4° in facing), and the model exceeded these variance levels in all but a few time steps.

The first experiment for this algorithm demonstrates the ability of the proposed method to calibrate the motion model parameters for a robot with little or no previous knowledge of the environment. The learning process is performed over the entire loop of hallway from the D-Wing data set. Note that the completion of a loop is not necessary for either the SLAM algorithm or the learning method, but merely serves to help illustrate the quality of the map at each EM iteration. The motion model is set initially with no systematic biases (mean zero noise), but high variances. Figure 6.1 shows the highest probability map produced at the end of the first run of EM. The resulting map has the right general shape, but in the top left area where the robot returns to its starting position there is a significant error in the map, resulting in double walls. A closeup of this region is shown in Figure 6.2. After three EM iterations, the model parameters are refined to the point where the SLAM algorithm successfully closes the loop without any blemishes in the map. A closeup of the same area is shown in Figure 6.3.

One concern that we had when learning a motion model was the possibility of over-fitting the specific trajectory that was supplied to the SLAM algorithm. We would like the learned parameters to be tuned to the properties of the robot and environment, but not the quirks of individual data collection runs, since it would be it would be inefficient and contrary to the spirit of SLAM to learn a new motion model with EM every time that the robot is redeployed. To verify this generality of the motion model, we used one run of the robot to learn the parameters in the same indoor environment as before. Then, using this

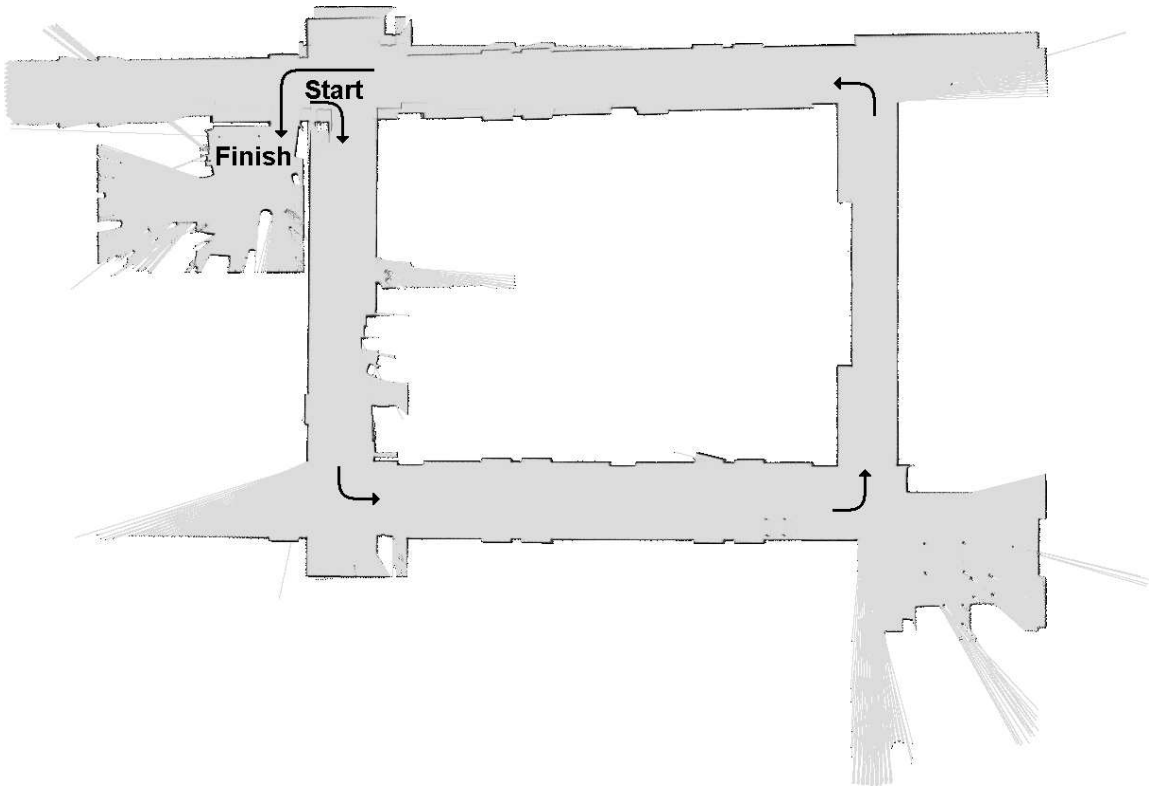

Figure 6.1: A complete loop of hallway, generated using a naive motion model. The robot starts at the top left and moves counterclockwise. Each pixel in this map represents 3cm in the environment. The total path length is approximately 60 meters. White areas are unexplored. Shades between gray and black indicate increasing probability of an obstacle.

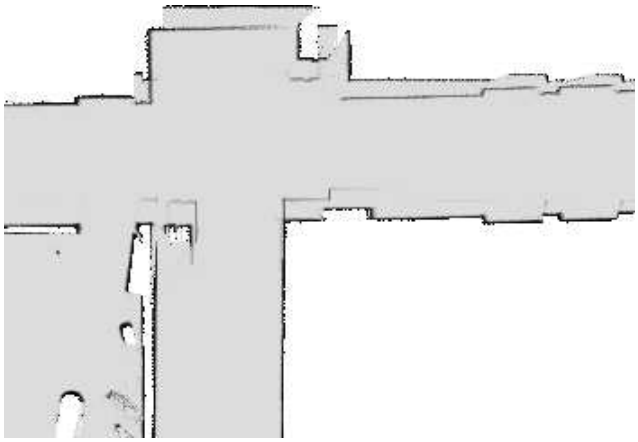

Figure 6.2: Close up of the area where the loop is closed, using the naive motion model. Double walls reflect an accumulated error of approximately one half meter over the path of the robot.

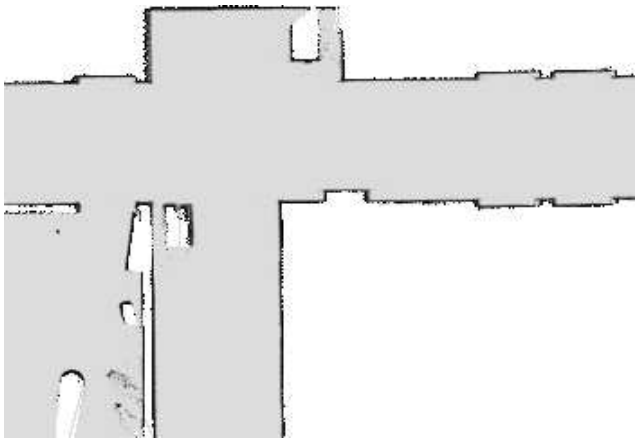

Figure 6.3: Close up of the same area as Figure 6.2, using the learned motion model learned by EM.

set of learned motion parameters, we had the robot remap the same environment using data collected from several days later. The resulting map shown in Figure 6.4 is the same high quality as if we had learned the motion model directly from the second trajectory itself.

A strong test of the robustness of our method is its ability to recover from a poor motion model. This is also important to the applicability of the method since changes in the environment or in the robot itself can cause the appropriate motion model to change. In this experiment, we used a set of data collected in an office environment to learn a good motion model. We then tested the model using a second set of data collected in the same area, but

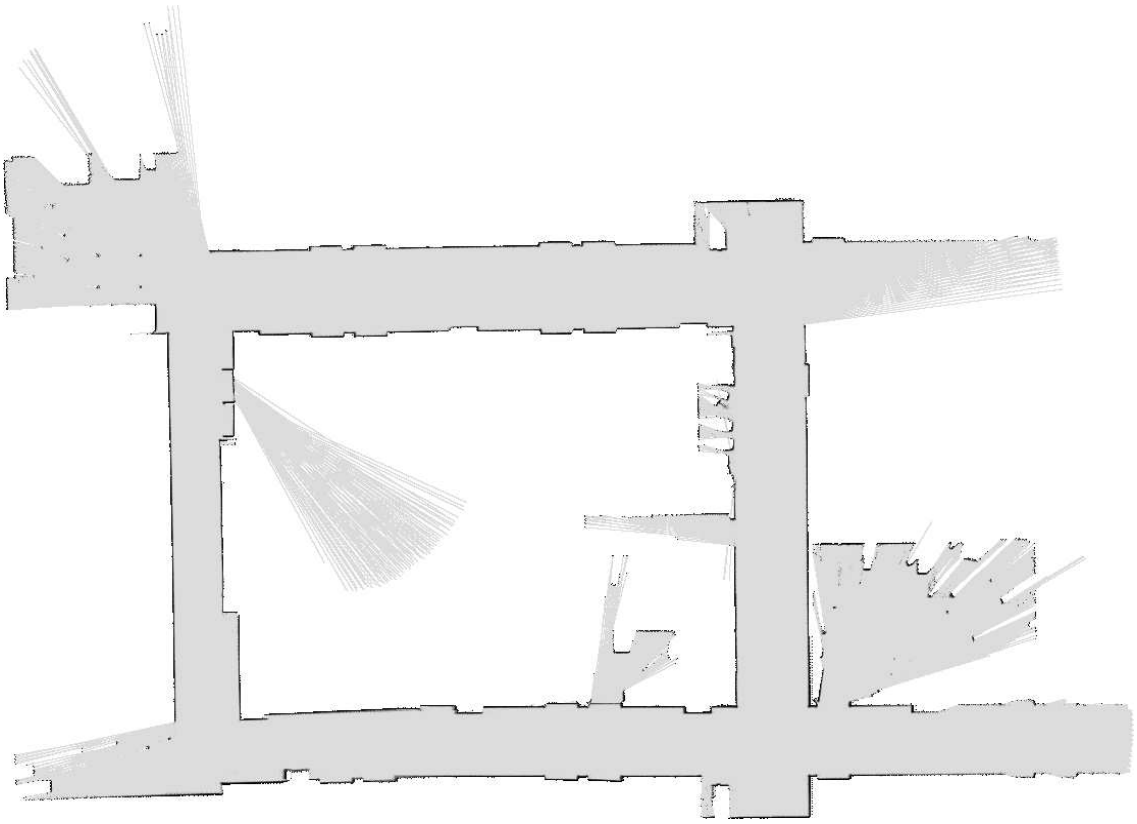

Figure 6.4: The map created using the motion model learned from one sensor log to on a different sensor log generated several days later.

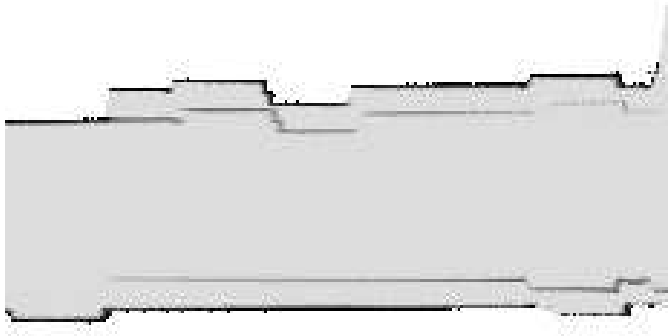

Figure 6.5: First iteration in correcting an inaccurate motion model: close up of loop closing.

with approximately a year of time separating the two data sets. When attempting to use the same model on the second data set, we quickly notice that the map produced by the SLAM algorithm is obviously flawed where it attempts to complete the loop (Figure 6.5). A year of use and some rough handling during shipping caused significant wear in the robot and changes in its behavior, resulting in an altered motion model. In the next iteration (Figure 6.6), the learned motion model can be seen to be improving the quality of the map as a result of increased accuracy. Figure 6.7 depicts the map from the next and final iteration, where the two ends of the loop are seamlessly aligned.

Figure 6.6: Second iteration in correcting an inaccurate motion model

Most of our results are visual or anecdotal, since the actual parameters would be fairly

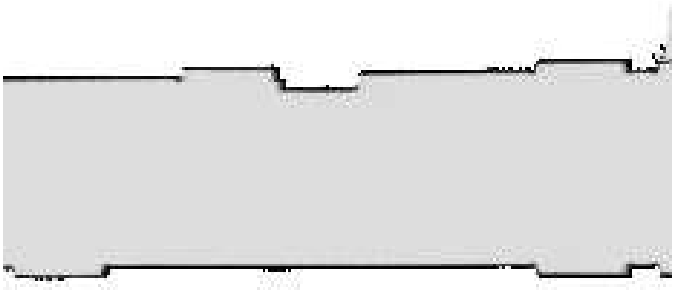

Figure 6.7: Final iteration in correcting an inaccurate motion model

meaningless to all but those very familiar with this model of robot. However, in this experiment the difference in variances is particularly telling. Predictably, the variances have all increased significantly over the course of the year, as wear on the robot has caused movements to become more erratic. The most significant of these is the σ_{C_t} term, which changes from $6.7\frac{cm}{rad}$ to $14.4\frac{cm}{rad}$, indicating significantly more erratic lateral shifts during turns, a result consistent with our observations of the robot in action.

We also performed an experiment on a smaller segment of sensor data. We wanted to use a section with a significant amount of both lateral motion and turning within its trajectory, so we chose an area consisting of two corners connected by one lateral stretch of hallway, for a total of approximately one quarter of a full sensor log. The initial model provided was the same naive model presented in the first experiment. We performed this experiment to verify the ability to learn a model with less information and to illustrate that traversing a loop is not necessary for accurate performance of the learning method. This experiment took ten iterations to converge while those based upon full sensor logs typically took less than five. However, the final motion model parameters upon convergence of EM were accurate enough to result in seamless mapping when the algorithm was presented with a full sensor log. The resulting maps are indistinguishable from those produced with models learned from full sensor logs, and as such, are not shown here.

Finally, we considered the possibility of using very large variances and a large number of particles as an alternative to learning a good model. The problem with this approach is that adequately covering the configuration space of the robot when the model parameters have high variance is quite difficult. Even with four times as many samples as our refined models, our naive model was unable to produce seamless maps.

Perhaps the most significant test of this method's ability to learn the motion model parameters for a given robot is to apply it to other robots, without any knowledge of that robot or its construction. Several sensor logs were obtained from other research groups, each recorded using their own robot. Using a mean zero, high variance noise model as our initial model, this EM algorithm was run on a short section (200 iterations, or approximately 5m of motion with turns) of the sensor log. Using no other alterations, the entire data log was successfully mapped. We present two of the larger data sets here.

Figure 6.8 shows a map of the convention center in Edmonton, Alberta where AAAI 2002 was held [1]. This sensor log, provided by Nick Roy, covers an area approximately 75m x 95m. The map shown was generated using 2000 particles, and used a motion model learned after six iterations of our EM algorithm.

Figure 6.9 is a map of another conference site, this one in Acapulco, Mexico, at the site of IJCAI 2001 [1]. The robot begins in the lower right hand corner, and observes a total area approximately 60m x 110m. The strange zig-zag pattern at the top are the posters on display, and the set of three small "rooms" on the right side of the image is the course for the robotic search and rescue competition. This map was made with 1500 particles, and spent eight iterations learning the motion model.

These results show that the method is capable of learning accurate motion models with very little user input. Beginning with a general, naive set of motion parameters, it is possible to refine the model to be significantly more accurate. In addition, this model can be generally applied to similar environments. Furthermore, when presented with an incorrect

Figure 6.8: A map of the conference center in Edmonton, Canada, where AAAI 2002 was held. The motion model used was learned without ever having seen the robot that collected the data.

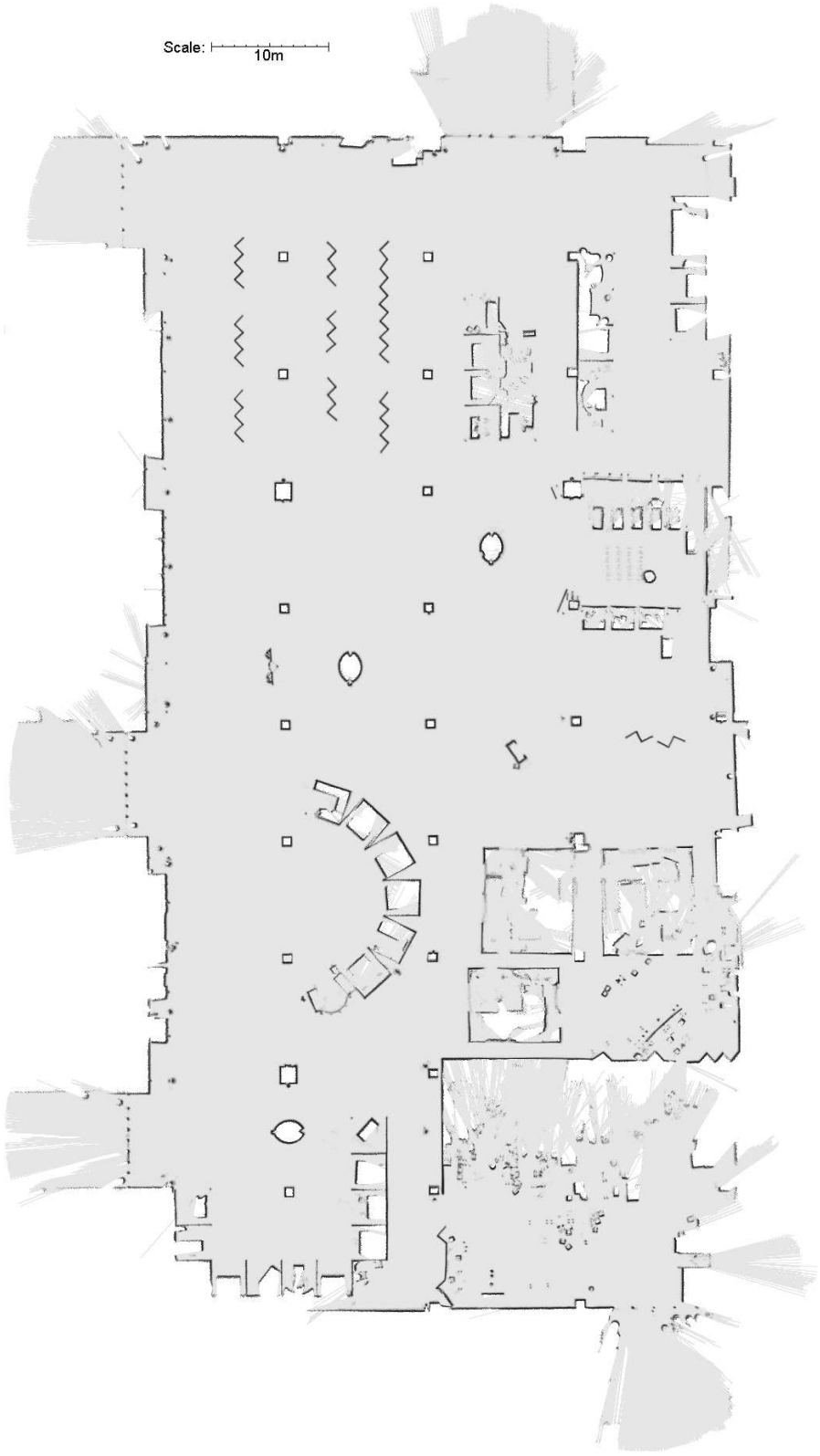

Figure 6.9: A map of the location of IJCAI 2001, in Acapulco, Mexico.

model, the proposed method quickly adapted, and was able to learn more appropriate parameters successfully. Finally, we demonstrated the power of this method to learn a good model that is applicable to a large area when presented with data from only a small piece of this area.

Chapter 7

Coalescence

Our research into particle filters is unusual, in that it specifically observes the ancestry of each particle, and keeps track of the relations and similarities between each particle. In doing so, we have observed an interesting phenomenon, called *coalescence*. As one traces the ancestry of the current particles back in time, the particle diversity decreases, indicating a general agreement among the current hypotheses about some earlier state. The point at which all current particles share a single ancestor is called the *point of coalescence*.

As can be expected, the number of iterations between the current generation and this point of coalescence is a function of the uncertainty present in the system. During periods of increased uncertainty this difference will grow, as the available particles spread out to cover a larger area of the state space. As more information is gathered, it is possible to resolve uncertainties in the past. Once this uncertainty is resolved, the point of coalescence will jump back towards the present iteration, indicating that all alternative hypotheses up until that point have a negligible probability.

7.1 Empirical Behavior of Coalescence

Let us examine the behavior of coalescence in a specific run of DP-SLAM. Figure 7.1 shows how the distribution of ancestor particles changes during the D-Wing experiment shown in Figure 4.10. The higher, red line indicates the number of ancestor particles maintained in the ancestry tree which were inserted at least five iterations earlier than the current iteration. The lower, green line plots the number of ancestor particles which were created at least ten iterations ago. As can be seen, these numbers vary significantly over the course of the experiment.

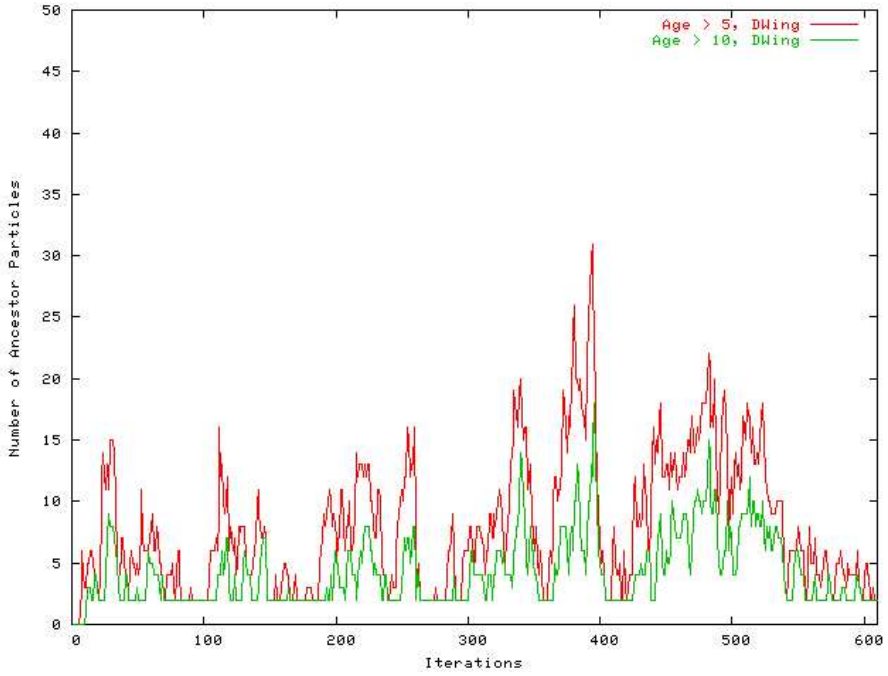

Figure 7.1: A graph of the coalescence behavior for DP-SLAM during the creation of the D-Wing map in Figure 4.10, using 3000 particles.

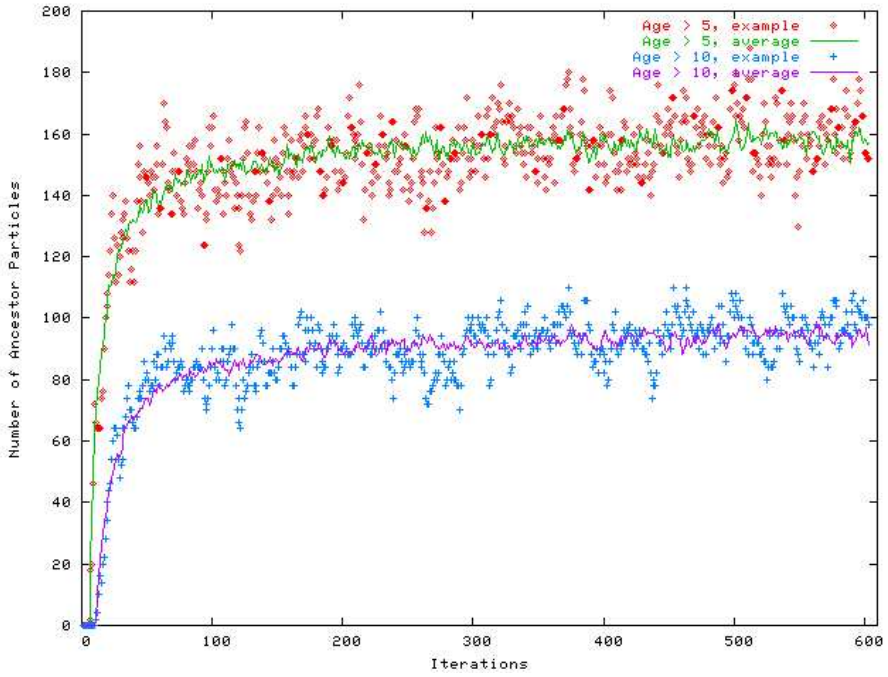

Figure 7.2: The coalescence behavior of a particle filter with 3000 particles, using completely uninformative data. All coalescence is purely the result of particle depletion.

There are a few sections of the graph which are particularly interesting. The initial uncertainty of the robot, as the first observations are being entered into the map, is apparent at the very start of the graph. Iterations 100 and 180 both mark the start of rises in the graph, and correspond to start of two new hallways: the right side hall, and the hall at the top of the map. As the robot first enters these hallways, there is very little information to disambiguate the robot's position, as there exist few features along the walls which the robot is able to observe very well. Similarly, the spike in the graph beginning around iteration 350 correlates with the robot entering the long section in the left hallway. The lack of doorways, or any other distinguishing feature, leaves the algorithm with very little information with which to confidently localize.

The final rise in uncertainty, from iterations 420 to 530, arises from a different source. In this area, the robot has closed the loop, and therefore has previously mapped this area. The robot is not lacking accurate map information to localize with. Instead, as the robot has not visited this section of the map recently, some of the data which was entered earlier is no longer valid, due to traffic in the halls. Most obviously, there is a door which previously was observed to be empty, and now the observations indicate that it is closed. These inconsistencies between the old map and the new observations have the effect of decreasing the algorithm's certainty of any particle, as no combination of map and pose fit the observations very well. This lack of confidence translates into a marked increase in the diversity of particles, which slowly returns to previous levels, as the map is updated with the new state of the world.

It is important to realize that this point of coalescence is not merely a side effect of random particle depletion. The point of coalescence is actually driven by the information present in the observations. To compare the results above to the behavior of an uninformed particle filter, a dummy particle filter was constructed. This dummy particle filter ignores the observations completely; each particle is automatically given the same weight as all of

the others. The number of particles and the resampling method stayed the same as those used in DP-SLAM for Figure 7.1. This dummy particle filter was run several times, to compare the coalescence behavior in DP-SLAM against the particle depletion which is observed in an uninformed particle filter. Figure 7.2 plots the results of one of these runs, shown by the points in the graph, as well as the average results over twenty different runs, indicated by the solid line. The higher set of traces represents the number of ancestor particles currently kept in the ancestry tree which were created more than five iterations ago. The lower trace indicates the number of ancestor particles older than ten iterations. As the figure shows, an uninformed particle filter quickly reaches significantly greater amounts of diversity in the population of ancestor particles, and maintains those high levels throughout the experiment. The amount of uncertainty in the system needs to be much higher before particle depletion begins to seriously affect the behavior of DP-SLAM.

7.2 Implications of Coalescence

This examination of coalescence in DP-SLAM provides important insight into the properties of DP-SLAM. Primarily, it establishes that the particle filter is maintaining the uncertainty in the proper area of the state space. This uncertainty is reduced as a natural consequence of the particle filter when future observations resolve the ambiguity. The reduction in the diversity of particles from earlier iterations is the result of information effectively propagating back in time, affecting the distributions at previous iterations, rather than result of random particle depletion.

A nice consequence of maintaining a relatively recent point of coalescence is a reduction in space complexity. All map updates made prior to the point of coalescence necessarily agree across all particles, allowing us to represent those sections compactly. As mentioned in previously in section 4.2.2, if the point of coalescence is defined as existing C iterations in the past, then the space complexity of maintaining distributed particle maps

is $O(M + ACP)$. This suggests that as the map increases in size, the memory required to represent the map approaches the size of maintaining only a single map.

So far, examination of coalescence has only been preliminary. DP-SLAM relies on the behavior of coalescence to be able to maintain multiple map hypotheses practically, yet it has not been thoroughly examined for other benefits. There remains a number of other potential applications for exploiting coalescence in particle filters. These possible uses could include variable particle numbers, adaptive resampling of past iterations, and more efficient data compression.

Chapter 8

Hierarchical SLAM

8.1 Drift

The DP-SLAM algorithm presented here provides a highly accurate and efficient method for building maps. However, for certain trajectories which cover a sufficient amount of distance before completing a cycle, the accuracy of the map can degrade. This problem of drift over large distances is a significant problem that is faced by essentially all current SLAM algorithms. Small errors can accumulate over several iterations, and while the resulting map may seem locally consistent, there could be large total errors, which become apparent after the robot closes a large loop. Due to an inability to represent the full joint probability distribution in sufficient detail, it becomes impossible to recover from these errors. In theory, drift can be avoided by some algorithms in situations where strong linear Gaussian assumptions hold [11]. In practice, it is hard to avoid drift, either as a consequence of violated assumptions or as a consequence of particle filtering. The best algorithms can only extend the distance that the robot travels before experiencing drift. Errors come from (at least) three sources: insufficient particle coverage, coarse precision, and resampling itself (particle depletion).

The first problem is a well known issue with particle filters. Given a finite number of particles, there will always be unsampled gaps in the particle coverage of the state space and the proximity to the true state can be as coarse as the size of these gaps. This is exacerbated by the fact that particle filters are often applied to high dimensional state spaces with Gaussian noise, making it impossible to cover unlikely (but still possible) events in the tails of distribution with high particle density. The second issue is coarse precision. This can occur as a result of explicit discretization through an occupancy grid, or implicit

discretization through the use of a sensor with finite precision. Coarse precision can make minor perturbations in the state appear identical from the perspective of the sensors and the particle weights. Finally, resampling itself can lead to drift by shifting a finite population of particles away from low probability regions of the state space. While this behavior of a particle filter is typically viewed as a desirable reallocation of computational resources, it can shift particles away from the true state in some cases.

The net effect of these errors is the gradual but inevitable accumulation of small errors resulting from failure to sample, differentiate, or remember a state vector that is sufficiently close to the true state. In practice, we have found that there exist large domains where high precision mapping is essentially impossible with any reasonable number of particles. Figure 8.1 demonstrates the result of DP-SLAM attempting to map Carnegie Mellon University's Wean Hall with 20,000 particles, requiring more than six hours of computation. As can been seen by the misalignment in the right hallway, the loop in this domain is large enough that existing particle diversity is insufficient to correct the inevitable small errors that occur.

8.2 Hierarchical SLAM

The DP-SLAM algorithm that we have constructed so far presents an approach to SLAM that reduces the asymptotic complexity per particle to that of pure localization. This is likely as low as can reasonably be expected and should allow for the use of large numbers of particles for mapping. However, the discussion of drift in the previous section underscores that the ability to use large numbers of particles may not be sufficient. If we accept that drift is an inevitable result of a sampling based SLAM algorithm, then we would like to have techniques that delay the onset of drift as long as possible. We therefore propose a hierarchical approach to SLAM that is capable of recognizing, representing, and recovering from drift.

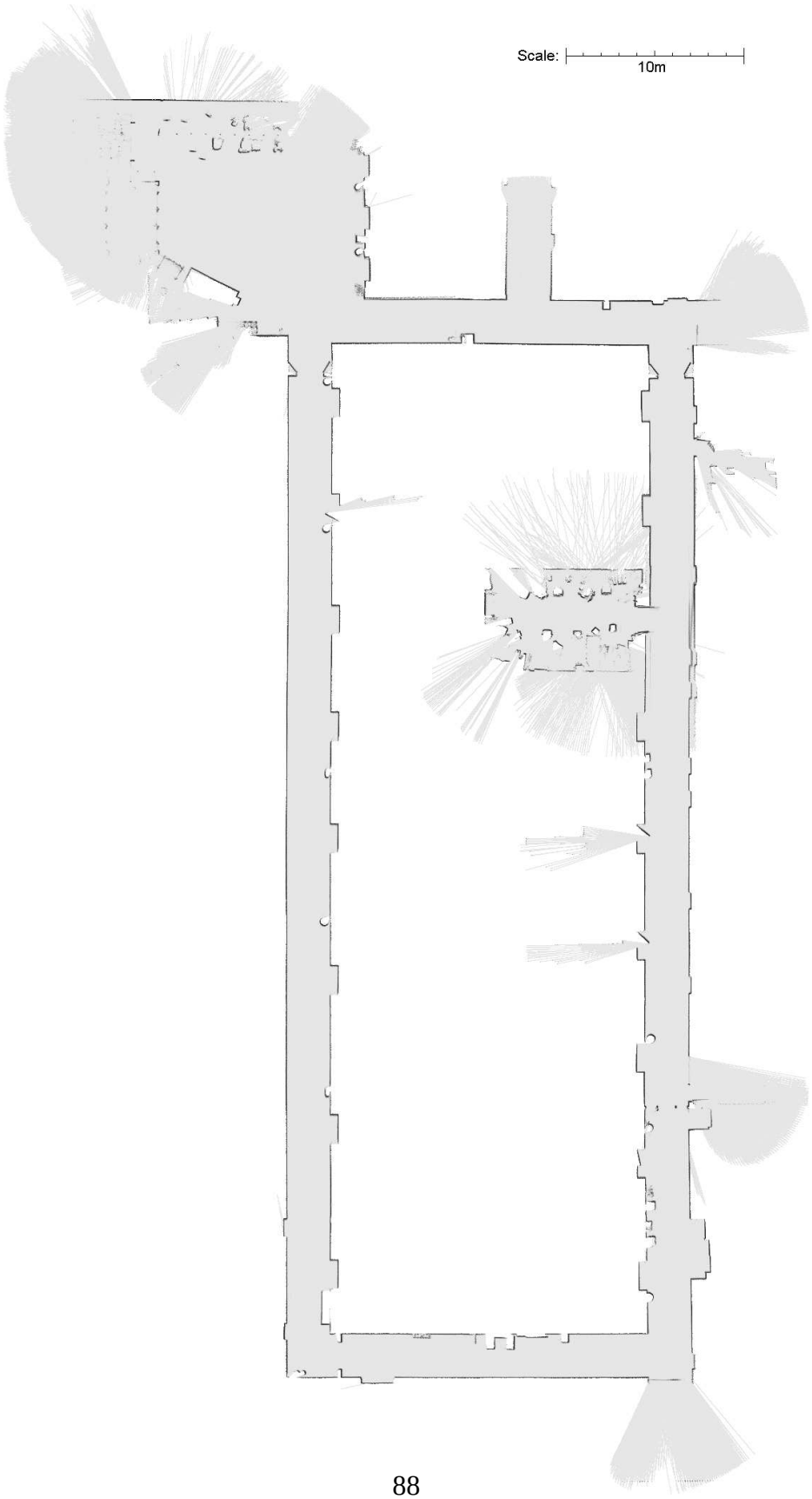

Figure 8.1: A map of CMU's Wean Hall, using a non-hierarchical implementation of DP-SLAM with 20,000 particles. There is a distinct error in the closing of the loop in the right hallway.

8.2.1 Related Work

Other methods have also attempted to preserve uncertainty for longer numbers of time steps. One approach seeks to delay the resampling step for several iterations, so as to address the total noise in a certain number of steps as one Gaussian with a larger variance [29, 30]. The information between resampling steps is not neglected, however. The weights used for resampling are the accumulation of the scores at the intervening steps. In general, there exist other look-ahead methods that can "peek" at future observations to use the information from later time steps to influence samples at a previous time step [35]. The HYMM approach[17] creates a number of small local maps, and seeks to combine them by overlaying them on a topological map.

A different way to interpret hierarchical SLAM is in terms of a hierarchical hidden Markov model framework [36]. In a hierarchical HMM, each node in the HMM has the potential to invoke sub-HMMs to produce a series of observations. The main difference is that in hierarchical HMMs, there is assumed to be a single process that can be represented in different ways. In our hierarchical SLAM approach, only the lowest level models a physical process, while higher levels model the errors in lower levels.

8.2.2 Hierarchical Algorithm

The basic idea is that the main sources of drift can be modeled as the cumulative effect of a sequence of random events. Through experimentation, we can quantify the expected amount of drift over a certain distance for a given algorithm, in much the same way that we create a probabilistic motion model for the noise in the robot's odometry. Since the total drift over a trajectory is assumed to be a summation of many small, largely independent sources of error, it can be well approximated by a Gaussian distribution.

If we view the act of completing a small map *segment* as a random process with noise, we can then apply a higher level filter to the output of the map segment process in an

attempt to track the underlying state more accurately. There are two benefits to this approach. First, it explicitly models and permits the correction of drift. Second, the coarser time granularity of the high level process implies fewer resampling steps and fewer opportunities for particle depletion. Thus, if we can model how much drift is expected to occur over a small section of the robot's trajectory, we can maintain this extra uncertainty longer, and resolve inaccuracies or ambiguities in the map in a natural fashion.

There are some special properties of the SLAM problem that make it particularly well suited to this approach. In the full generality of an arbitrary tracking problem, one should view drift as a problem that affects entire trajectories through state space and the complete belief state at any time. Sampling the space of drifts would then require sampling perturbations to the entire state vector. In this fully general case, the benefit of the hierarchical view would be unclear, as the end result would be quite similar to adding additional noise to the low level process.

In SLAM, we can make two assumptions that simplify things. The first is that the robot state vector is highly correlated with the remaining state variables, and the second is that we have access to a low level mapping procedure with moderate accuracy and local consistency. Under these assumptions, the effects of drift on low level maps can be accurately approximated by perturbations to the endpoints of the robot trajectory used to construct a low level map. By sampling drift only at endpoints, we will fail to sample some of the internal structure that is possible in drifts, e.g., we will fail to distinguish between a linear drift and a spiral pattern with the same endpoints. However, the existence of significant, complicated drift patterns within a map segment would violate our assumption of moderate accuracy and local consistency within our low level mapper.

To achieve a hierarchical approach to SLAM, we use a standard SLAM algorithm for the low level mapping process. The input to the low level algorithm is a short portion of the robot's trajectory, along with the associated observations. This SLAM process runs

normally, with no alterations. The only difference is that the output that we use from this algorithm is not only a distribution over maps, but also a distribution over robot trajectories.

We can treat these distributions over trajectories as a distribution over motions in the higher level SLAM process, to which additional noise from drift is added. This allows us to use the output from each of our small mapping efforts as the input for a new SLAM process, working at a much higher level of time granularity. Since the sampled trajectory is treated as a single, atomic motion, this defines the placement of the associated observations. The observation model at the high level is then just the collection of observations that were made at each step along this trajectory. The high level algorithm otherwise behaves in much the same way as any other SLAM method, and will combine a large series of iterations into a larger map.

For the high level SLAM process, we need to be careful to avoid double counting evidence. Each low level mapping process runs as an independent process initialized with an empty map. The distribution over trajectories returned by the low level mapping process incorporates the effects of the observations used by the low level mapper. To avoid double counting, the high level SLAM process can only weigh the match between the new observations and the existing high level maps. In other words, *all* of the observations for a single high level motion step (single low level trajectory) must be evaluated against the high level map, before any of those observations are used to update the map. We summarize the high level SLAM loop for each high level particle as follows:

1. Sample a high level SLAM state (high level map and robot state).
2. Perturb the sampled robot state by adding random drift.
3. Sample a low level trajectory from the distribution over trajectories returned by the low level SLAM process.
4. Compute a high level weight by evaluating the trajectory and robot observations against the sampled high level map, starting from the perturbed robot state.

5. Update the high level map based upon the new observations.

In practice this can give a much greater improvement in accuracy over simply doubling the resources allocated to a single level SLAM algorithm because the high level is able to model and recover from errors much longer than would be otherwise possible with only a single particle filter. In our implementations we have used DP-SLAM at both levels of the hierarchy. However, there is reason to believe that this approach could be applied to any other sampling-based SLAM method just as effectively. We also implemented this idea with only one level of hierarchy, but multiple levels could provide additional robustness. We felt that the size of the domains on which we tested did not warrant any further levels.

The computational complexity of hierarchical SLAM requires a slightly different analysis than the ones we have performed already. Since we use a linear implementation of DP-SLAM at both levels, each level itself can run in $O(AP)$ time. However, the A and P terms are different for the two levels, so the complexity is really $O(A_L P_L + A_H P_H)$. We can assume $P_H \cong P_L \cong P$. Unfortunately, it appears that $A_H \gg A_L$. If we assume the low level of the hierarchy runs at a frequency k times greater than the high level, then $A_H \cong kA_L$. Therefore, the high level is only run $\frac{1}{k}$ as often, implying that the total run time for the high level, compared to the run time of the low level, is $\frac{1}{k}(kA_L P) = A_L P$. Thus the total complexity for both levels is still $O(AP)$.

8.3 Implementation and Empirical Results

To demonstrate the benefits of hierarchical SLAM, we chose to use a data log of Carnegie Mellon University's Wean Hall, shown in Figure 8.2. In this domain, the robot travels approximately 220m before returning to an earlier position. This data set has previously been found to be very difficult for non-hierarchical SLAM algorithms. Linear DP-SLAM was successfully able to map the environment, but required at least 120,000 particles and

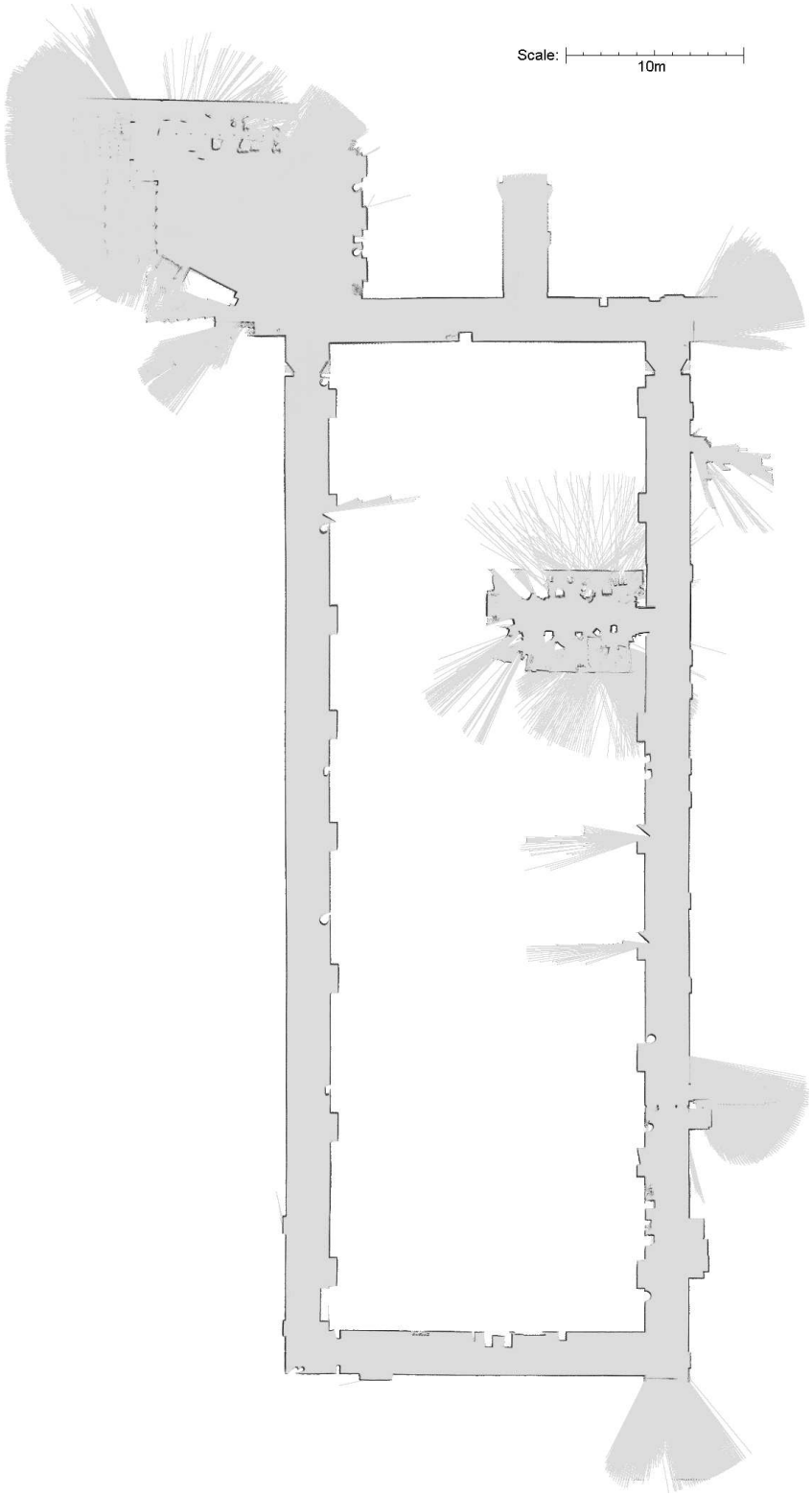

Figure 8.2: CMU's Wean Hall at 4cm resolution, using hierarchical SLAM.

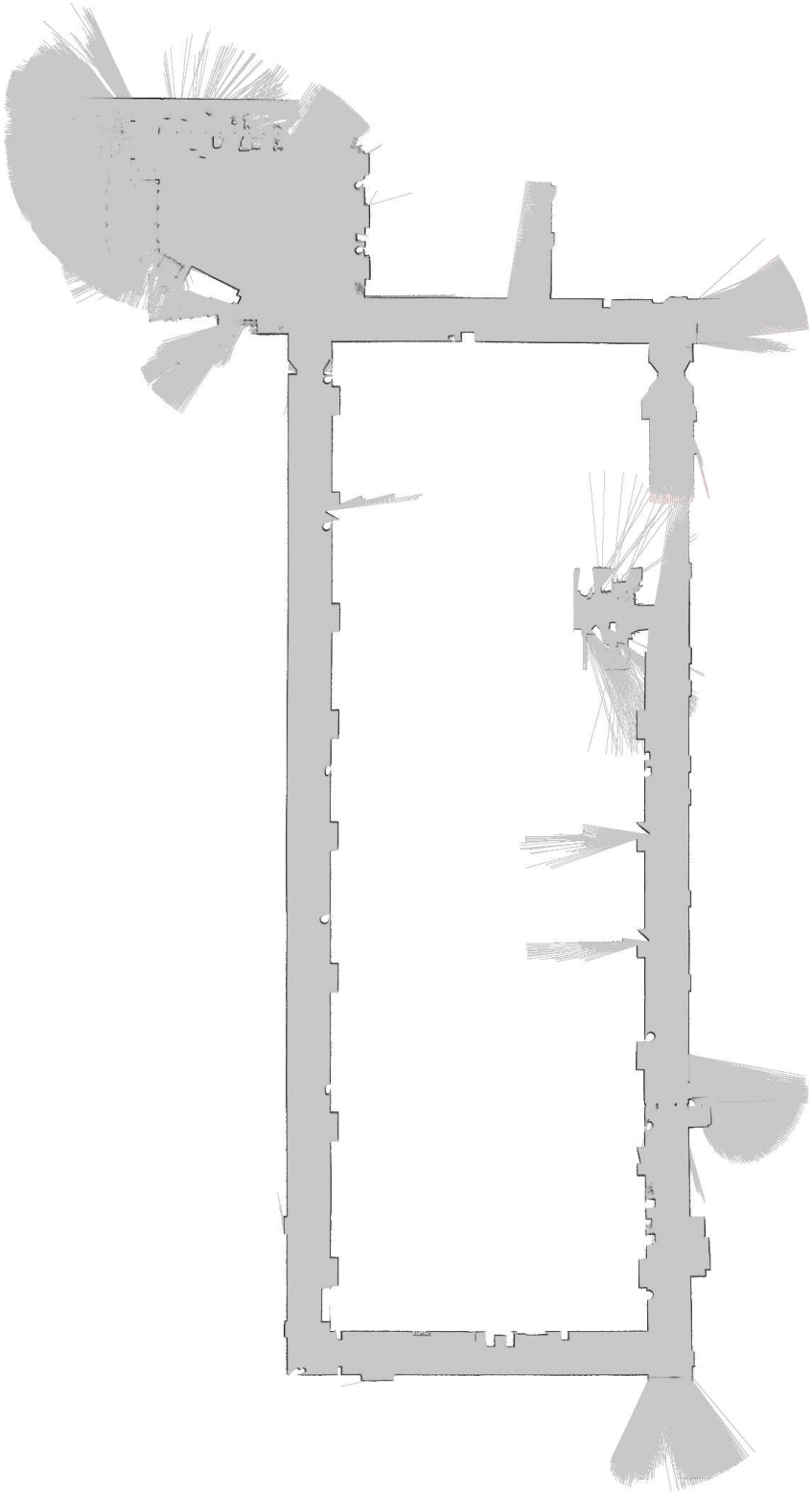

Figure 8.3: A depiction of the current uncertainty in the map, shortly before the non-hierarchical approach attempts to complete the loop. Pink areas indicate sections of the map where the given map has no observations, but alternative hypotheses do have entries.

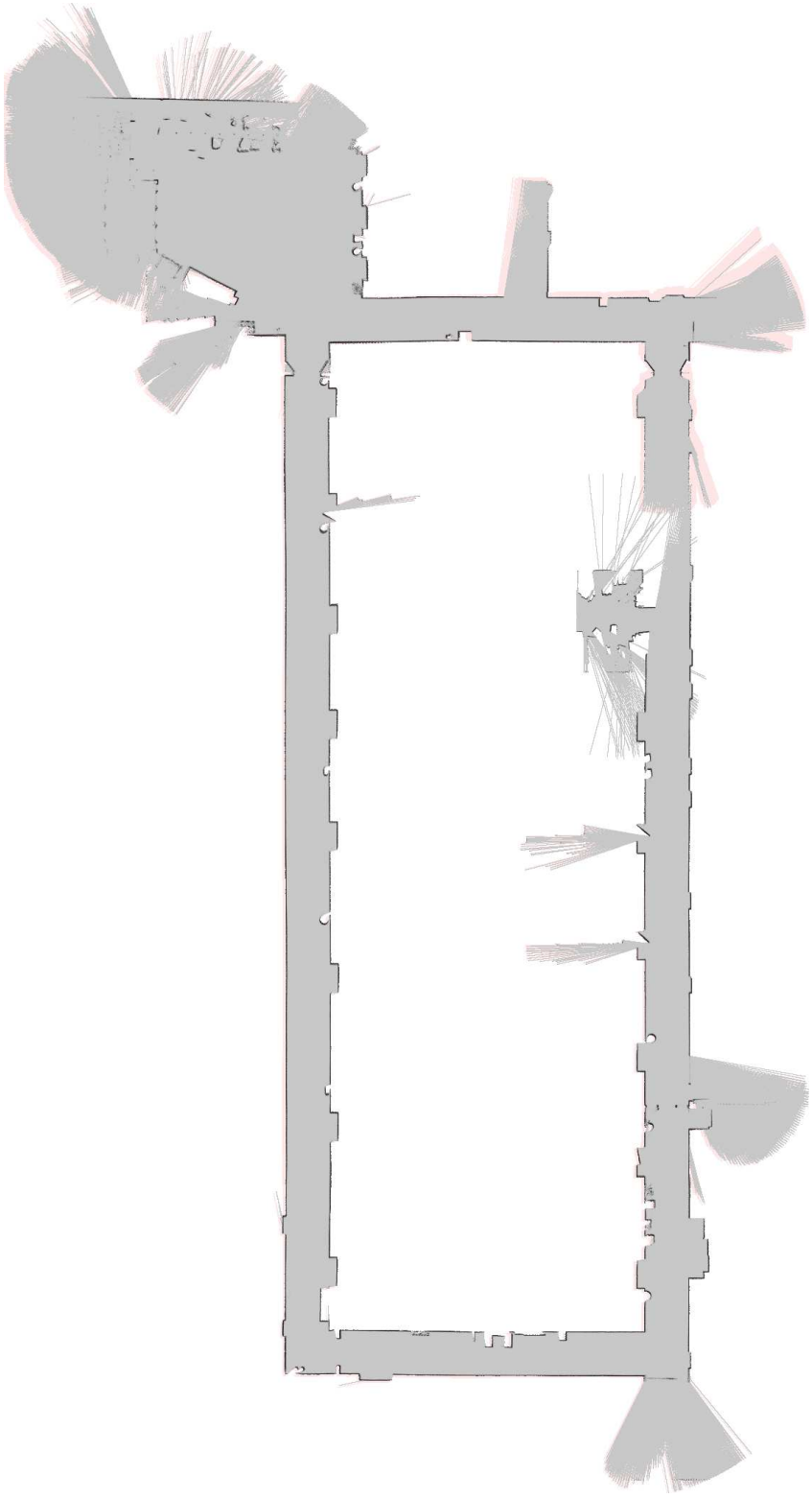

Figure 8.4: The amount of uncertainty in the map (once again shown in pink) is much greater for the hierarchical approach.

took more than 42 hours on a 3.2 GHz processor. This is significantly more particles than the few thousand typically accepted as a reasonable number.

When we ran a hierarchical version of DP-SLAM on the data, each low level SLAM process was run for 75 iterations, with an average motion of 12cm for each time step. This process was able to map the building successfully with significantly fewer particles, 2000 for the low level and 3000 for the high level, and it took only 4 hours and 53 minutes. This extreme difference in particle counts and computation time demonstrates the great improvement that can be realized with the hierarchical approach. (The Wean Hall dataset has been mapped successfully before at low resolution using a non-hierarchical approach with run time per iteration that grows with the number of iterations. [29].)

The reason that a non-hierarchical method has such a difficult time mapping this data is the extreme longevity of the uncertainty. Given the large size of the loop, small ambiguities in the beginning of the map are not resolved for several thousand iterations. Non-hierarchical DP-SLAM requires a tremendous number of particle to maintain this early particle diversity long enough. As Figure 8.3 indicates, there is little uncertainty in the map as the loop nears completion.

Compare this to the amount of uncertainty maintained by hierarchical DP-SLAM, as shown in Figure 8.4. Shortly before completing the loop, the high level particle filter can be seen to maintain multiple hypotheses all the way back to the beginning of the trajectory. In fact, at this point, the point of coalescence for the high level is at the very first iteration. For the non-hierarchical approach to maintain this same amount of uncertainty, it would need to have a point of coalescence nearly 2000 iterations ago. In fact, its point of coalescence at this stage is close to twenty iterations ago. As the robot completes the loop, the uncertainty in the hierarchical method is nearly eliminated, as the additional information from revisiting the starting area is sufficient to resolve the remaining ambiguities.

Hierarchical DP-SLAM is also useful for simpler domains, such as the C-Wing data

set used earlier. The non-hierarchical version of DP-SLAM required 11,000 particles to reliably map this domain, and even with the linear time implementation, it required 690 minutes. The using the hierarchical version required 2000 particles for both the high and low levels, and was able to complete the map in 123 minutes. Thus the significant reduction in particle counts for the hierarchical implementation provides dramatic improvements in running time, even in situations that would otherwise be possible map without the hierarchy.

8.4 Extensions of Hierarchal SLAM

So far, we have only extended the hierarchical framework for SLAM to two levels, as this has been sufficient for our data sets. For the paths traveled during these experiments, the distance is small enough between loop closing events that there is not yet a significant amount of drift at the high level. However, for sufficiently large domains, it is reasonable to expect that such drift could occur. In that case, one could extend the hierarchical framework further, and add an additional layer to the hierarchy. Naturally, it has been difficult for us to empirically test the feasibility of this idea without artificially altering the data set or significantly hampering the underlying SLAM method.

This idea of hierarchical SLAM does not need to be restricted to being used solely with DP-SLAM. The sources of drift outlined above are inherent in any sampling based SLAM method. Likewise, none of the elements of the hierarchical structure we create are specific to DP-SLAM; since the hierarchical structure solely focuses on engineering inputs and outputs, this method should be equally effective when used in conjunction with any other SLAM method.

Chapter 9

Practical Improvements

9.1 Culling

At each iteration of DP-SLAM, we generate many more particles than we keep. Evaluating a particle requires line tracing 181 laser casts. However, most particles will have significantly lower probability than others and this can be discovered before they are fully evaluated. Using a technique we call *particle culling* we divide the laser scans into k partitions, and evaluate our set of particles in k passes. At each pass, the particle's pose is evaluated against the map for those $\frac{181}{k}$ laser casts, and that total probability is multiplied by the previous probability for that particle. The highest probability particle can be identified, and a single scan through the set of particles can remove those particles which have a significantly lower probability than the current best particle. In this manner, particles with very low probability of getting resampled are never fully evaluated. In practice, this leads to large reduction in the number of laser casts that are traced through the grid.

The trick to our culling method is to remove as many particles as possible, while keeping all of the particles which would normally have a high probability of being resampled. Let us examine the probability of any given particle being resampled. For the sake of simplicity, we can assume that their weights are normalized. Given a weighted set of P particles, the probability of choosing a specific particle p_n, is equal to that particle's normalized weight w_n. For resampling, we make P independent selections, with replacement, so the chance of choosing a given particle is now $1-(1-w_n)^P$. For purposes of culling, we would like to remove those particles which are least likely to be resampled, so we can set some sort of threshold, and any particle whose normalized weight is less than that threshold can be eliminated. If we were to ignore the particle weights, the probability of choosing

any specific particle would be $\frac{1}{P}$. Requiring that each maintained particle be more likely than this uninformed choice gives us a threshold value of $\frac{1}{P}$. In practice, this has provided us with good results. This number can of course be altered, depending on how aggressive the user is about culling.

Expanding on this idea of early heuristic evaluation, we can also preprocess the entire set of particles for obvious poor poses. To scan through the different particles quickly, we initially evaluate them solely on the evidence of a partial line trace. This is a much shorter line trace, centered around the laser endpoints, and is only a few grid squares long in either direction from the endpoint. Since this heuristic does not include all of the information present in a full line trace, it is unable to detect objects which should obstruct the laser early on in the scan, and therefore it is a somewhat coarse evaluation. However, it is able to quickly determine which particles do not have a significant portion of their endpoints line up with well established obstacles, which is sufficient to identify a large number of the poor particles in a very fast manner.

When combined together, these two heuristics significantly reduce the total number of particles that need to be completely evaluated, and give an average practical speed up on the order of $20\times$. Extensive comparative empirical analysis of these methods has convinced us that no particles which are eliminated early would have otherwise lasted for more than one iteration. Thus the approximation does not degrade the accuracy of our algorithm.

9.2 Important Parameters

Perhaps one of the most obfuscated and annoying aspects of implementing another researcher's algorithm is attempting to infer the values of important parameters. These "magic knobs" represent important values in the algorithm which were arrived at through experimentation or intuition, and are often difficult for other researchers to understand, much less duplicate. Hopefully to make this somewhat clearer for DP-SLAM, we will

briefly review the most important of these parameters, and discuss their role as well as reasonable values for them.

9.2.1 Observation Model

Within the robot's observation model are two important parameters. The first of these is the laser variance. This number expresses the expected amount of error that should be present in any one of the laser scans, when compared to the robot's map, assuming that both the map and the current robot position are both correct. Notice that this is not purely the amount of noise present in the sensor. Although this is a major contributing factor, it is also affected by noise and clutter in the environment (which can lead to 'blurry' object boundaries) as well as the the scale of the map itself, since the precision of the observation model is influenced by the resolution of the map. The main effect that this parameter has on the algorithm is to affect the resampling stage. A smaller variance creates stronger distinctions between particles' weights, and thus makes it less likely for many particles to be resampled each iteration. A good value that we tend to use for most domains at a resolution of 3cm tends be a standard deviation of 4cm.

The other important parameter in the observation model is the maximum allowed error. This is a term which limits the total amount of error that a single observation can have. Seen a little differently, this term represents a certain amount of background noise, past which point any reading is equally likely. This helps to model several seemingly aberrant events that can occur in the environment, such as semi-transparent or semi-reflective surfaces, or objects with small holes or sharp boundaries. Since the probability of a single robot pose is the product of the probability of each laser cast, this term is important to ensure that a single poor observation (with probability near zero) does not completely eliminate an otherwise good pose. In practice, we use a value of $e^{\frac{50}{2\sigma^2}}$, where σ is the standard deviation of the laser's noise discussed above, as the lower bound for the probability that any single

laser cast can be assigned.

9.2.2 Motion Model

The most obvious parameters relating to the motion model of a robot are the motion parameters themselves. This includes a mean motion, usually derived from the robot's odometry, and a set of variances, which describe the size and shape of a multidimensional Gaussian distribution about that mean. A full discussion of the motion model that is used in our implementations, and how the values can be obtained, is described in great detail in the earlier chapter on proposal distributions.

One of the important issues in implementing a SLAM algorithm that is rarely discussed is the speed of the robot, or more precisely, the speed of sensory input into the the algorithm. Due to the crucial issue of particle depletion, it should be obvious that the separation between iterations should be based on distance traveled, rather than time elapsed. In particular, if the robot is not moving at all, continuing to model "motion" in the robot is pointless. We have found that the appropriate distance between obtaining observations about the environment should be close enough together as to allow for significant overlap between one observation and the next, yet far enough apart so that there are minimal resampling errors. While previous work has come up with methods for resampling occasionally while still using all of the incoming data [29], we have found that the simpler approach of merely filtering out the intermediate information is sufficient for our purposes. The intervals that we generally use are a minimum of 10cm of lateral motion, or 0.04 radians of angular motion.

9.2.3 Map Parameters

There exist several map parameters that can be controlled by the user. The first of these is a practical limitation on the size of the map allowed. The map width and map height describe the size of environment, and, as such, define the size of many internal data structures, and

by extension have a profound impact on the amount of memory used. This parameter naturally can vary greatly, depending on the domain that the robot will be expected to cover.

Another map parameter that should be acknowledged is the map prior. This is the probability of occupancy which is assigned to any previously unobserved grid square. This value can of course be changed according to the clutter in the environment. Recall that the probability of laser being obstructed by a given grid square is determined by an exponential distribution $P_c(x, \rho) = 1 - e^{\frac{-x}{\rho}}$, where x is the distance that the laser traveled through that square. As described earlier, we use a gamma distribution as a conjugate prior, with a shape parameter of 1. This conjugate prior has the same practical effect on each unobserved grid square as if there had been a previous observation with parameters $d = 0.04$m and $s = 0.005$.

The last map parameter is the scale of the map. This dictates how much area in the real world is represented by a single grid square in the map representation. The map resolution can affect the speed, accuracy and memory requirements of the algorithm. Coarser resolutions will of course tend to be faster and require less memory, but can fail to represent important features in the environment accurately, and can increase the amount of drift. However, if the resolution is too fine, this can lead to problems with accuracy as well; due to the spread of the different rays cast out from the laser range finder, a fine enough grid can fail to have significant overlap from one time step to the next, even when the robot's relative motion is small. In our experience, we have found that resolutions of 3-5cm to a side for each grid square to be best, though in certain domains, 10cm can be sufficiently precise.

9.2.4 Hierarchical Parameters

The hierarchical SLAM structure creates multiple SLAM processes running alongside each other. The lowest level is the basic SLAM algorithm, working unperturbed. However, the higher levels are working with slightly different data, and as such, require a different laser noise model. While the actual noise in the laser has not changed, a large number of factors about the map have. With a rigid trajectory passed up from the lower level, there is less room for minor perturbations, and certain amount of assumed drift. In addition, the number of observations has scaled, and as such, the difference between very similar poses scales. All of this is included in the high level laser variance, which needs to be correspondingly larger. Empirical results for our domains show that using a standard deviation of 7cm at the higher level works well.

As the observation model changes in hierarchical SLAM, so does the motion model. No longer are we modeling the noise in the odometry readings, but instead the noise in the lower level's mapping accuracy. This "motion" model is assumed to be Gaussian, and evenly distributed about the lateral axes. The specific values for these variances are highly mutable, affected by the specific SLAM algorithm used at the low level, and amount of resources used, as well as elements from the robot or the environment. This values could either be altered by hand, or learned in a similar manner as the odometric motion model at the lower level.

Beyond modeling parameters, hierarchical SLAM also needs to decide the duration of the lower level mapping process. Typically expressed in numbers of iterations, this number should reflect how long the low level mapper can reliably be trusted to create maps with no serious misalignments or errors. This is another number that varies considerably, but in our experiments, we tend to keep it between 75-150 iterations

Chapter 10

Summary of 2-D DP-SLAM

Through the development of DP-SLAM, we have made significant progress towards creating an accurate, principled SLAM algorithm, capable of efficiently maintaining multiple map hypotheses. At the core of this solution is the concept of maintaining an *ancestry tree*, in order to better represent the differences between the separate hypotheses. The information contained in this ancestry tree is sufficient to allow us represent the map data in a very compact manner, without having to maintain any information multiple times, and without a need for map copying. This core idea of *distributed particle mapping* is the key concept that allows practical use of multiple map hypotheses, where earlier attempts were bogged down.

This distributed particle representation has a very nice implementation, which allows for a linear time asymptotic complexity, in both the number of particles and the size of the observations. By expanding those grid squares which are to be observed in a given iteration into easily accessible observation caches, we can reduce the access time of each grid square to constant time, and achieve a time complexity which is identical to that of pure localization with a single map. The co-development of methods such as culling, to remove poor particles before they are fully evaluated, allow us to run DP-SLAM with sufficiently large numbers of particles in real time to map many interesting environments.

The stochastic map formulation described here provides a rigorous and flexible representation of the environment, and effectively models the behavior of the laser through various media and surfaces. By properly modeling inconsistent behavior within certain areas, we are not constrained to treating as many different events as “background noise”, and are better able to distinguish the sub-grid square resolution in the robot’s pose.

Process	Naive	DP-SLAM	Linear DP-SLAM	Hierarchical
Localize	$O(AP)$	$O(AP^2 \log P)$	$O(AP)$	$O(AP)$
Insert	$O(AP)$	$O(AP^2 \log P)$	$O(AP)$	$O(AP)$
Delete	N/A	$O(AP \log P)$	$O(AP)$	$O(AP)$
Merge	N/A	$O(AP^2 \log P)$	$O(AP)$	$O(AP)$
Resample	$O(MP)$	$O(P)$	$O(P)$	$O(P)$

Table 10.1: Summary of Computational Complexity

The last major contribution to two dimensional SLAM was the development of hierarchical SLAM. Even the most precise mapping methods can accumulate drift as the robot travels increasingly long distances. Hierarchical SLAM is a principled method of recognizing this drift, and representing the drift in a way that makes it possible to easily recover from the tiny, inescapable errors that build up. Using hierarchical SLAM on the largest of our environments, we able to map significantly larger loops than would otherwise be possible. Even those domains that were mapped previously can be done so in less time and with fewer particles through the use of hierarchical SLAM.

All of this together puts us close to achieving our goals in two dimensions. A principled approach to maintaining multiple maps is possible, producing very detailed, accurate maps which are produced regardless of the path of the robot's trajectory; loop closing is used as a measure of accuracy, and not as a tool to provide accuracy. All of this is possible in real time, using reasonable computing resources. To this effect, research has made great progress towards considering two dimensional SLAM solved. However, there are still improvements left to achieve for the two dimensional case, and much work to be done for effective three dimensional mapping, the ultimate goal of SLAM.

Chapter 11

3-D SLAM

11.1 Preliminary 3-D SLAM Work

Existing SLAM methods tend to be restricted to planar cases, where the map produced is a two dimensional cross section of the world, and robot motion is restricted to motion within this plane. Not only does this constraint prohibit use in many interesting environments, effectively exiling the robot to the floor of a building, it is also insufficient to represent objects and obstacles which may be displaced from the plane, such as table tops or stairwells.

While some methods exist for three dimensional motion, they tend to represent the world in terms of a few sparse, point-sized landmarks. These maps, while useful for localization, and possibly for navigation, give very little information about the presence of objects in the world. Carnegie Mellon University's Mine Mapping project is a notable exception, which built volumetric three dimensional maps using a series of laser range finders set at different angles [37]. Using a combined method of both local and global scan matching techniques, a two dimensional occupancy grid is created. Using the corresponding trajectory for the robot, the remaining, three dimensional, data is filled in to create the volumetric maps. Thus a three dimensional map is constructed, while motion in the third dimension is ignored.

The DP-SLAM architecture, as it exists now, has only been applied to two dimensional mapping. However, the restriction to planar motion is unrealistic in most applications of mobile robots. Wheeled robots traveling across uneven terrain, and underwater autonomous vehicles (UAVs) can move with six degrees of freedom, three lateral and three angular. For the robot to operate in this environment, we not only need to track these three new degrees of motion (roll, pitch and height), but also maintain a three dimensional

representation of the environment.

This expansion of the problem presents two types of challenges, technical and dimensional. The technical problems are mainly issues of sensing. Previous sensors used were well suited for a two dimensional world, but are unable to make observations outside of the plane. In particular, odometry is unable to detect any motion in the three new degrees of freedom. Also, laser range finders, which are so popular for two dimensional mapping, are typically a planar sensor. While three dimensional laser range finders exist, they are prohibitively slow, and do not allow the robot to move while making an observation. Therefore, 3-D SLAM not only needs a new measure of estimated motion for the proposal distribution of the particle filter, but also a new sensor for the primary observations of the environment.

Dimensional challenges are problems of scale. As new dimensions are added to the problem, the resources needed to deal with SLAM grow exponentially, so that merely extending previous methods is infeasible on any computer architecture we might expect to see in the near future. One of these challenges deals with space requirements. Two dimensional maps are large data structures, and proper maintenance of multiple map hypotheses can already take on the order or gigabytes of memory for reasonably sized environments. Adding a third dimension to these maps in a straightforward manner would, of course, increase the memory requirements from N^2 to N^3, where N is the linear size of the environment, quickly placing the memory requirements outside the realm of feasibility. The other challenge that needs to be addressed here is one of time complexity. Robot motion in three dimensions allows for six degrees of freedom, which is twice as many parameters to track as in the planar case. This means not only that more particles are likely to be needed by the particle filter, but that the sensor also needs to gather more information, e.g., by making more independent observations of the environment to disambiguate the robot's pose. Maintaining these extra particles, comparing these extra observations with the map,

and adding each of the new observations to the map all take significantly more time. In our preliminary work, we attempt to address all of these challenges, but focus particularly on decreasing the complexity of map updates.

11.2 Technical Issues

As can be expected, extending SLAM to handle 3-D maps does not merely increase the dimensionality of the problem. The same set of sensors that were effective for motion within a plane are no longer sufficient to give useful information in a three dimensional environment. Odometry is no longer sufficient to capture the motion of the robot, as shaft encoders cannot measure roll, tilt, or changes in elevation. Likewise, a planar range sensor, such as the laser range finders which are popular in 2-D SLAM, is of limited use in three dimensions. Even slight motion out of the plane will cause a new observation to misalign with previous scans.

Fortunately, both of these issues can be addressed using existing methods in computer vision. Motion estimates can be obtained through maintaining a "visual odometer". In addition, the field of stereo vision has provided a wealth of methods for getting useful 3-D sensor data, if one is willing to accept the associated running time for the time being. We take a closer look now at the implications of using these methods for SLAM.

11.2.1 Proposal Distribution / Motion Estimation

One of the most basic steps in almost all existing SLAM methods is the generation of a proposal distribution. Based upon the robot's last position, an initial estimate of the robot's current position is generated, based upon either control inputs or a secondary sensory input. The use of odometry for this step is almost universal, but as has been already noted, odometry is insufficient for capturing three dimensional motion.

One option is to augment the standard shaft encoders with a set of inclinometers. These

sensors are able to directly measure the robot's pitch and roll. However, they are at least as noisy as shaft encoders, and can be expensive. Also, the change in elevation can only be indirectly observed.

Another option is the use of accelerometers, in order to measure displacement by observing changes in velocity over time. This has the nice property that the robot is no longer required to have wheels, or even travel over land. However, the problem of excessive noise is even greater for these sensors, and good accelerometers remain costly.

Visual odometry methods, such as Lowe's Scale Invariant Feature Tracker (SIFT), provide a very nice, low cost alternative approach to tracking incremental motion[38]. By identifying a number of points of interest in a series of stereo image pairs, SIFT solves the data association problem in order to track the relative motion of the camera with respect to the world.

SIFT keypoints have many nice properties, which have generated some interest in using them as features for a landmark-based SLAM method. However, these features of interest are not always found at the exact same world coordinate position in each frame, due to uncertainty in the SIFT algorithm. In addition, SIFT keypoints are usually short lived, and usually are not observable from substantially different perspectives. However, they still perform very nicely as a dead-reckoning system, and are perfect for use a proposal distribution in other SLAM methods.

11.2.2 Observation Dependence

As stated before, sensors which make range measurements directly, such as laser range finders, are difficult to use with three dimensional motion. Instead, we use stereo correspondence methods to create a three dimensional depth observation. This works well in conjunction with SIFT, since the method for extracting keypoints in SIFT automatically rejects any potential keypoints which are not sufficiently distinctive for a given image. This

virtually eliminates the potential for false matches, which is the primary source of error in stereo algorithms. Therefore, when the cameras are properly calibrated, the noise in one method can be assumed to be independent from noise in the other.

Unfortunately the independence assumption is not true for different depth measurements within a single stereo pair matching. Stereo algorithms work by taking a section of the left camera image, and comparing it with different sections of the right image. However, a single pixel is not enough to create reliable matches between the images, so an entire window around each pixel is used to compute the correspondences. These windows overlap across neighboring pixels, so that the error in one depth estimate is no longer independent of the depth estimates nearby. However, these dependencies become weaker the further the pixels are from each other in the image plane. In fact, for some stereo methods, if a pixel is not within the stereo window of another pixel, those two pixels are not directly dependent, and assuming their independence introduces negligible bias.[1] Therefore, in using stereo depth estimates for localization, we use only every nth row and column, where n is the size of the stereo window. We then treat this subset of readings as independent, and compute the probability of each measurement separately, given the map and the robot pose, and take their product as the total (unnormalized) weight for a particle.

It is important to note that map is merely a record of the observations, which is used an assumption of the future behavior of the sensor in that area. For this purpose, we can make a good approximation of the empirical opacities by treating the noise for each pixel matching as independent when building the map. Therefore, we expect that it is more informative to use every pixel matching to update the map.

[1] Independence along the vertical axis is common in stereo vision algorithms. However, horizontal dependence may exist for some algorithms, such as dynamic programming methods.

11.3 Data Explosion

One of the most difficult problems of expanding SLAM into three dimensional environments is the simple problem of dimensionality. By allowing movement outside of a plane, the scope of the problem increases dramatically. Not only does the map representation move from two dimensional to three, but tracking now expands from three degrees of freedom to six. This implies that a naive extension of two dimensional work in SLAM will not suffice. The vast amount of resources needed, both in terms of time and space, would make implementation infeasible. More clever algorithms are needed to manage these new dimensions before the more subtle difficulties of three dimensional SLAM can be identified and addressed.

11.3.1 3-D Mapping with Voxels

The most natural extension to the grid-based map representation of DP-SLAM is the use of voxels. These metric cubes can take on the same properties of stochastic grid squares, while representing volumetric space. Like the previous implementations of grid squares, these voxels are small in scale, on the order of a few centimeters. The problem is that a three dimensional map is equivalent in size to a large number of two dimensional maps stacked on top of each other, one for each height increment. For a map resolution of centimeters, this can easily mean the voxel map is several hundred or even thousand times the size of a grid-based map. Given the large amounts of memory needed to maintain a single planar map for DP-SLAM, the memory requirements for a complete voxel representation of the world would be infeasible.

What I propose instead is to maintain a data structure which behaves very much like a full voxel representation, while on average requiring only the equivalent space of a map which is just a few voxels high. This is achieved by condensing a series of empty voxels, which all share the same (x, y) coordinates in the world, into a single interval.

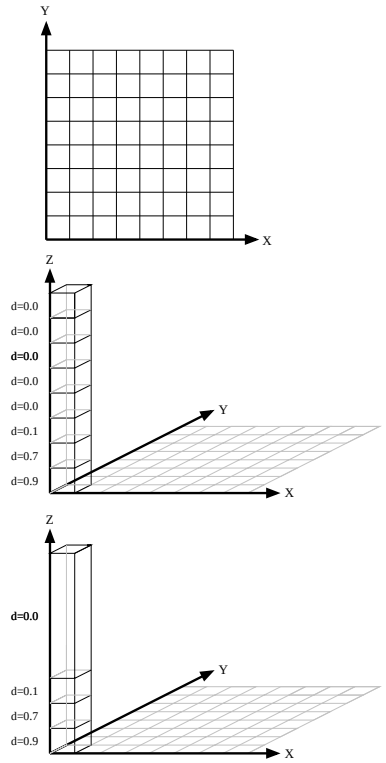

Figure 11.1: Left: a planar grid-based map. Center: a 3-D voxel-based map. The density coefficients for each voxel are listed beside them. Right: A 3-D map, condensing adjacent empty voxels in the same column into a single interval.

Consider a single, planar grid for the map representation. Let each entry in this grid correspond not to a single grid square, but instead contain the entire column of voxels that exist at this (x, y) location. If the entries of this column are indexed with an array, the map representation degenerates back into a full voxel representation. However, we can do better than that. In nearly all environments, the total observable space of the world is overwhelmingly dominated by empty space. Most applications for the map data, including localization, treat this empty space as all equivalent; there is very little to distinguish one part of empty space from another. Furthermore, observable empty space tends to come in long stretches. Therefore, it is reasonable to consider condensing large intervals of these empty voxels into a single entry, describing the scope of the entire interval. Since these intervals are dynamic while the map is still being built, the set of all voxels in a given column are stored in a balanced tree, keyed on z position, with empty space using the lower end of the interval as the z value.

The resulting map representation requires significantly less space to store than a complete voxel representation, due to the compact representation of empty space. For a column of space hundreds of voxels high which crosses but a few object boundaries, the space required to represent the observed voxels in this column could easily to less than a dozen entries. Furthermore, any unobserved voxels are not even included in the set.

These benefits of course come at a price. First, there is a slight increase in the running time needed to access a voxel. The search through the set of observations requires $logN$ time, where N is the number of entries for the column. Luckily, the same argument which implies a good space reduction also ensures that this time increase will be small, since both are dependent on the number of column entries.

The other drawback of this method is that it becomes necessary to introduce an approximation to the SLAM process. Since each empty interval is stored as a single entry, the entire interval necessarily has to be homogeneous. This means that unlike the previ-

ously described approach to stochastic occupancy in DP-SLAM, we are unable to keep a meaningful observation odometer for any portion of the interval over empty space. That is to say, we are not storing how much that area has been observed, and correspondingly, we do not have a measurement of how confident we are in the designation of that area as unoccupied. Therefore, if an object is ever detected in any one of the voxels along the interval, the new occupied value for that voxel will only be an approximation of the actual observed occupancy.

This all describes how to build a single three dimensional map efficiently. We still need to address the core property of DP-SLAM, describing how this can be used to maintain multiple maps for each of the distinct particles efficiently. Recall that in two dimensions, we represent the distinctions between the different map hypotheses at each grid square, by storing up to P different opacity values in the observation vector. Given the use of intervals to represent empty space, it is difficult to extend this concept efficiently to the three dimensional case, by maintaining the distinctions between maps at each voxel. Each particle can have a different start and end point for a given empty interval. Thus it either becomes necessary to maintain these empty intervals only at areas where all particles agree, or else a more complicated method would be needed to describe the combined intervals.

Our approach takes a step back from the entire issue, and instead maintains the distinctions between particles at a slightly coarser level. For each (x, y) location, we let each particle maintain its own column. As each particle adds a new observation to the map, it can create its own set of column entries. Just like two dimensional versions of the distributed particle map, any section which is not observed by the current observation can be seamlessly inherited from the parent's hypothesis, simply by not creating a new entry for that particle. When localization realizes that the particle has no entry for the desired voxel, it will continue back up the ancestry to the parent of the particle.

This mechanism can be exploited for empty space as well. We are now representing all

empty space identically, since we do not keep a record of how much we have observed an empty voxel. Therefore, if a voxel has already been designated as empty, any further observations that continue to indicate that the voxel is empty do not require updating that voxel. This drastically reduces the number of ancestor particles which create new, updated entries for that empty voxel, further reducing the space necessary to store the three dimensional maps.

This three dimensional approach can be viewed as a direct extension of two dimensional DP-SLAM. Looking at it this way, the map is still a two dimensional grid, maintaining multiple observations at each grid square. The main difference in the map representation is that the notion of an observation is changing; the data stored by each particle is now representing an entire column of space. Consequently, each access to the map returns a complex set of data for the given particle, rather than a single opacity. For a given particle to retrieve the information for a specific voxel at location (x, y, z), we first need to access the observation for this particle at grid square (x, y). Then we need to search over the column of data in that observation to return the specific voxel (or empty interval) at height z.

11.3.2 Localization

Localization using one of these three dimensional maps is very similar to localization for DP-SLAM with a two dimensional map. The two main differences – the subsampling of the observation and the use of a column of data at each grid square – have already been described.

The stereo vision depth map is first subsampled to create a set of individual range measurements, which are treated as possessing independent noise. For each particle, this set of observations is treated in much the same way as any other range sensor: a ray is traced through the map from the origin of the sensor to the endpoint of the observation. For

each (x, y) position crossed by this three dimensional ray trace, DP-SLAM retrieves the corresponding data for the given particle at that grid square. As described earlier, this data consists of a set of individual voxels and intervals of empty space; finding the appropriate z value within the column requires a search over the set of data. Otherwise, the trace for each observation is performed in a manner identical to two dimensional DP-SLAM. The probability of each measurement, given the map and the robot's pose, is computed using an appropriate observation model. As before, the probability of each particle is simply the product of the probabilities of each individual observation.

11.3.3 Map Updates

Another issue of dimensionality concerns the sensor directly. Using a stereo method for the range observations introduces its own jump in complexity. Planar SLAM algorithms primarily use laser range finders, or similar sensors, which have on the order of 180 sensor readings comprising a single observation. Stereo, on the other hand, can have millions of depth estimates for a single observation. This is significantly more than the 180^2 that would produced by a three dimensional range finder of the same observation density as the current planar LRFs. Stereo produces much denser information, and it would be unfortunate not to use as much of that data as we can to reduce the uncertainty within the map. However, performing millions of line traces through the map is a daunting task. A more efficient method is needed if using all of the data is to be feasible.

As described earlier, the problem is already reduced for the case of localization. Approximating an independence assumption has forced us into using a subsample of the observation. This is not necessarily a serious loss of data; use of the subsampled data will likely give a very close approximation to the particle weights obtained by using the entire observation. However, for updating the map, it is important to have dense updates, so that a future observation can line up with the map, without falling along gaps. Therefore,

subsampling is not the preferred option for mapping.

The solution for efficient map updates comes from the very density of the observation data. For a typical camera and lens, the number of pixels across a single row is sufficiently large, so that even out to a distance of 8m, adjacent pixels correspond to points no more than 1cm apart. Therefore, for a reasonable map resolution of 3cm to a voxel side, any voxel within the field of view of the camera will be observed by multiple pixels, unless it is occluded by an object closer to the camera.

This density of observations means that the observations do not need to be considered a set of rays to be traced anymore, but instead can be viewed as a single polyhedron of empty space, resembling a pyramid originating from the camera, with an irregular base, corresponding to the observed objects in the frame. If we can describe the exterior of this polyhedron in terms of voxels, the actual map updates can be made quickly, similar to drawing a polyhedron using raster lines in graphics. Even better, the boundaries of the polyhedron represent the upper and lower boundaries of an empty interval, which is the natural form for our map representation.

Figure 11.2 illustrates an example of one of these polyhedrons of observations. In the top image we see a simple cross section of the camera viewing the environment. The area observed is denoted by the grey area. The first two boundaries describe the two sides of a triangle, and are the perimeter of the camera's field of view. The observations are limited to a certain distance, creating the curved boundary on the left. The black section represents the observed terrain, and serves as another, irregular section of the polyhedron's perimeter. The final section of the perimeter is the occlusion boundary created by the obstruction jutting up out of the ground. Taken together, we can see the enclosing perimeter of the observation in the middle figure. This completely describes the area of the environment which is currently observed as empty. As described above, we can easily find the height at which each column in the map will intersect with the perimeter of the observation. This

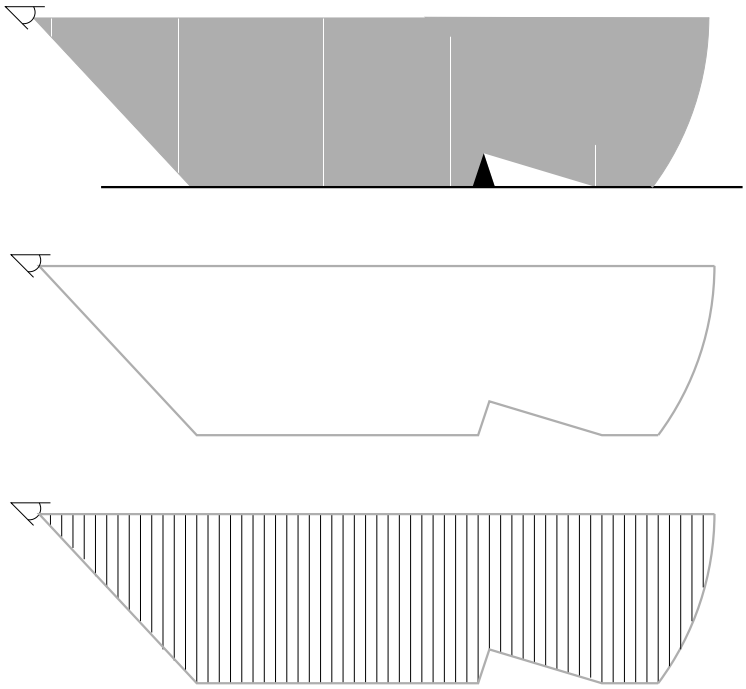

Figure 11.2: A cross section of a single observation, illustrating how the area being observed forms a polyhedron. The perimeter of this polyhedron is all that is needed for updating the map.

allows us to describe the new intervals of empty space, according to a given observation, in an efficient manner.

To describe this polyhedron in terms of voxels, some ray tracing is required, though significantly less than a full trace per pixel. We begin by performing a full ray trace for each pixel on the first row of the image. This will form a single flat side of the polyhedron from which to build. Subsequently, the next rows of pixels are added to the description. For these later pixels, however, it is only necessary to trace a small portion of the corresponding ray. Since we are only concerned with tracing the perimeter of the observation polyhedron, we only need to perform that section of the line trace which extends beyond the surrounding polyhedron. Our partial line traces therefore proceed as follows:

For each pixel, p, we first consider all of the pixels which are adjacent to p within the image plane, paying attention to the one whose corresponding distance from the camera is the smallest. Starting from the endpoint of p, we trace a ray *backwards* towards the camera,

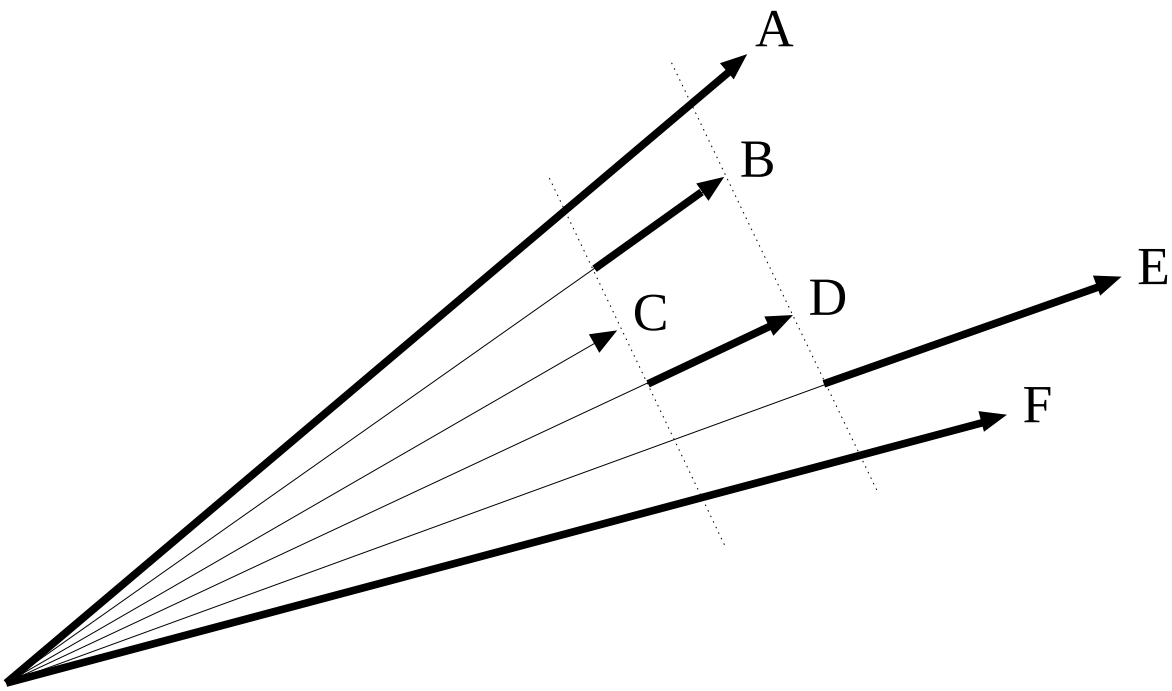

Figure 11.3: An example of a partial line trace for a dense sensor reading. The darker lines indicate which portion of the individual rays would need to be traced in order to ensure coverage of the perimeter of the observation polyhedron.

until we are closer to the camera than any of the adjacent pixels. This ensures that any portion of the line trace for p which could possibly be on the perimeter of the observation polyhedron has been covered. The remaining section of the line trace which has not been covered is all ensured to be on the interior of the polyhedron, since the adjacent pixels all project out past that portion of the line trace.

This point is illustrated in Figure 11.3. Here, the observation has again been reduced to a two dimensional cross section for ease of illustration. A number of rays, A through F, are emitted from a sensor, and are assumed to be dense enough to be treated together as a single polyhedron. Lines A and F are both on the far edges of the polyhedron, and thus need to perform a complete line trace, as indicated by the thicker line, in order to define the far edges of the the polyhedron. Lines B and D both consider their neighboring scans, and find that C is the shortest adjacent line. Therefore, they only have to trace that portion of their ray trace which extends beyond C. The rest of the ray trace would always

have another scan on all sides of it, which extends further out. This guarantees that the remaining portion of the scan will not be on the perimeter of the polyhedron, and thus does not need to be traced. The line C itself is a local minimum; all surrounding lines extend further than C. This means that only the endpoint of line C could possibly be exposed to the exterior, and therefore no portion of that line needs to be traced.

Along each partial line trace, we place a marker at each voxel we passed through, indicating its possible status as a portion of the perimeter. After all pixels are visited, this collection of perimeter markers can be combined to create a set of empty intervals, corresponding to the interior of the polyhedron. These empty intervals are then easy to incorporate into the map as a new set of updates. Occupied observations, by definition, will only occur at the endpoint of a ray, and can easily be inserted into the map separately.

We have described how to update a column of space efficiently. However, we have not specified where the data that we modify comes from. Recall that the map cache maintains only a single pointer at each (x, y) location. Therefore, each data entry for a grid square in the distributed particle map needs to maintain the entire column of data for a given particle, not just what was observed by that specific particle. Otherwise, for each gap in the column, corresponding to unobserved space, any particle attempting to access this data would need to search across all of its ancestors, to see if any of them had observed that precise voxel. This would eliminate the constant time access which the map cache was built for. This implies that each time that a particle updates a voxel in any given (x, y) position, the entire column of information inherited by that particle would need to be copied over into a new entry, before any updates could be made. As we will see when we analyze the complexity of this algorithm, this copying step is not a limiting factor on the running time of entire method.

11.4 Computation Complexity

3-D DP-SLAM offers a nice extension of the framework of DP-SLAM into a three dimensional representation of the world. Much of the same structure and methods are preserved, which helps to provide an efficient running time. However, there are a few changes in the asymptotic limits.

One of the important differences in the complexity of the localization stage is the subsampling of the observations. Instead of using the entire set of observations, we use a distinct subset of the range measurements. Let us denote the area covered by this subset of data as A_L. Using the map cache data structure, we can preserve constant time access to each (x, y) position in the map. However, this only provides the column data for each grid square. In order to retrieve the appropriate voxel information for a given height, we need to search across this column. If we define the maximum number of entries in one of these columns as V, each search is limited by $O(\log V)$. This provides us with an upper bound for localization of $O(A_L P \log V)$. For any natural environment, it is reasonable to expect $O(\log V) \cong O(1)$, as the number of object boundaries in any given column will be small. Even for man made environments, the probability that a flat surface, such as a wall, is perfectly parallel with the z axis is negligible.

An important additional consideration is the cost of building the map cache. While we perform line traces only on a subset of the data, the total area covered by all particles with this subset is likely to be much closer to $O(A)$. However, since the particle distinctions are maintained at the level of the grid, the map cache only needs to be as large as the projection of A onto the grid, A_π. The data stored at each grid square in the three dimensional map is still no more than one (complex) observation per particle. Therefore, building the map cache in the three dimensional case is no more difficult than the two dimensional case; this step can be completed in $O(A_\pi P)$ time.

Deletions from the map are very easy to perform. Using the list of map updates at

each ancestor particle, the observations can be deleted directly from the grid. Since all observations made by a single particle are removed at the same time, the information within the column data set can be ignored. Given that the projection of the area observed at each time step onto the grid is A_π, each iteration will only add the potential for $O(A_\pi P)$ deletions.

Merging data from the collapse of a branch in the ancestry is simple for the same reason. Recall that the expensive portion of a merge involves comparing the list of updated map locations for a parent particle with that of the child. In the three dimensional map, these lists of updates point to columns of data stored in the (x, y) grid. Therefore, the potential number of observations inserted at any given iteration is $O(A_\pi P)$. When comparing the lists, each entire column from the child is inserted into the parent's list as an entire unit. Similarly, an entry from the parent which is being replaced by the child's update can be removed as a single unit. Therefore, the amortized analysis of merges provides a complexity of $O(A_\pi P)$.

The first step in updating the map is for each particle to create a new observation in the (x, y) grid for each grid square within the projection of the current observation. As currently described, this requires the particle to make a copy of all of the data in this column inherited from its parent. Given a column of size V, this will require $O(A_\pi PV)$ time.

To update the map for a given particle, the surface of the current observation is traced. Recording each voxel of the surface does not require any access to the map; this record of surface is kept separately for the time being. Therefore, each step in tracing the surface can be performed in constant time. If we denote the size of the surface of the observation polyhedron as A_S, the time required to trace the surface for each particle is $O(A_S P)$.

In the process of tracing the surface of the observation polyhedron, each column is assigned a list of z positions, indicating where the surface insects with this column. As

we are tracing the surface of the observed area in order from top to bottom, this list can be assumed to already be sorted by z value. These points of intersection define the intervals over which the map will be updated with an empty observation. The data within the column is stored in a balanced tree, keyed on the z value. Therefore, the items in the column can be traversed in order, in $O(V)$ time. Given these two ordered lists, a comparison to find which items in the column need to be updated can be performed in $O(V + S)$, where S is the number of intersections for that column. This update needs to be performed for each grid square in A_π, and for each particle, providing a complexity of $O(A_\pi P(V + S))$. Note that $A_\pi S$ is in fact the size of the surface of the observation, A_S. Therefore, this bound can be simplified to $O(A_S P + A_\pi PV)$, which is the same complexity required for the two preliminary steps of the map update.

In analyzing the total complexity, we arrive at $O(A_L P \log V + A_P S + A_\pi PV)$. This is a complex bound to place on the running time, but one that accurately describes the different influences on the running time. The amount of resampling used for localization, the complexity of the observations, and the density of small objects in the environment can each define the dominant term for the upper bound. There are a couple of nice properties of this bound, however. First, the complexity is still linear in the number of particles used. This aspect is particularly important, as the number particles needed to track the robot's motion in three dimensions is typically greater than the two dimensional case. Second, none of the area terms include the entire size of observation. Using this method of storing the z dimension in columns of data allows us to avoid including the tremendous area covered by the sensor in our complexity bound.

11.5 Initial Results

To demonstrate the effectiveness of the proposed methods, we performed a series of experiments. All of these experiments were performed on stereo data collected from the NASA

Ames Marscape, using the K9 rover. In these experiments, the robot observed a total area of approximately 15m by 17m, with a total height observed of 3m. The data collected were then run offline, in simulation of the actual exploration, with a map resolution of 3cm.

First, after giving such a pessimistic description of the memory requirements for a voxel-based map, we need to prove that our proposed map representation can actually be implemented within reasonable memory requirements. To represent this area with a full voxel map would require an estimated 15Gb of memory, which was more than we have available. Implemented with the proposed map structure, the total memory used was only 511Mb.

Our second experiment was to demonstrate the effectiveness of updating the map using the perimeter of the observed polyhedron. During an experimental run of the data, we recorded the time required to update the map for a single particle. Updating the map with a full set of line traces took an average of nearly an hour. By using only the perimeter of the polyhedron, the map update time was reduced to an average of 2.5 seconds.

Our final empirical result was to demonstrate the early results of voxel mapping. In these experiments, a significant amount of distortion was observed in the stereo matches, for which we were unable to correct. Therefore, given poor input data, we would not expect very good accuracy for the maps. However, the results are still impressive. In the run, the robot entered the Marscape, and traversed the perimeter of a crater bed. Figure 11.5 shows the resulting map as the robot completes a loop, and is able to see the entrance to the crater again. The map displayed is a topographical view, with lighter areas representing higher elevations of the terrain. Several bulges can be seen in the center of the crater, which correspond to rocks and debris. Paying particular attention to the rocks at the top of the map, near where the loop is closed, it can be seen that when these rocks are observed from a new angle, the map places them at a very similar location, but due to warping effects of distorted stereo, the rocks do present a slight “double image”. This sensor error would

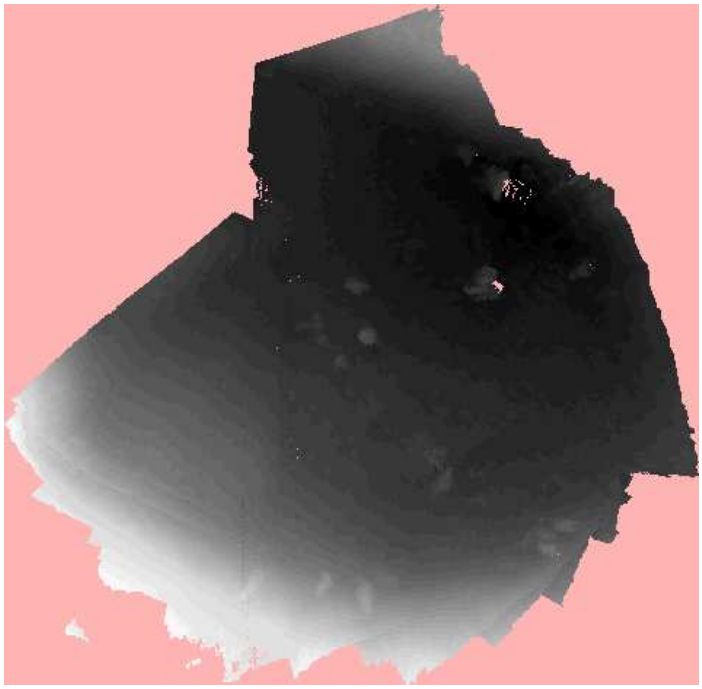

Figure 11.4: Initial mapping results from traversing part of the way around the perimeter of a 15m radius crater. The map shown is a topographical view, with lighter areas representing higher elevations.

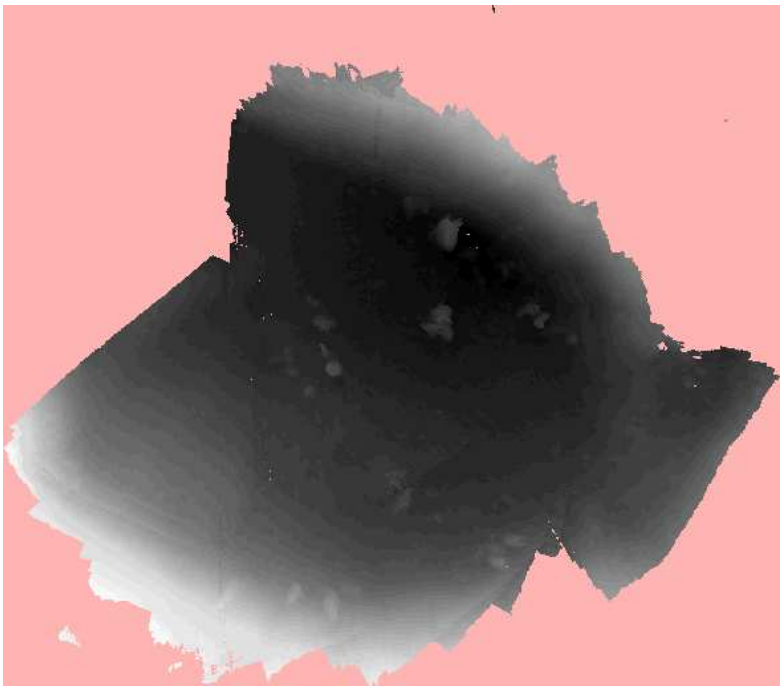

Figure 11.5: The resulting topographical map after completing the loop of the perimeter of the crater.

need to be corrected in future experiments. However, these results are encouraging, in that even with poor data, the resultant map is reasonable and relatively consistent.

Chapter 12

Future Directions

12.1 Alternate Sensors

With the exception of the preliminary three dimensional experiments in the previous chapter, DP-SLAM has focused primarily on a laser range finder as our primary sensor. This is a nice tool, in that it can make very accurate range measurements, and can reliably sense objects out to distances of 10-30m, depending on the particular model used.

However, laser range finders do have certain drawbacks. A laser necessarily traces a single, nearly one dimensional line through the world. This means that a laser range finder, emitting several of these laser beams along a plane, is necessarily a two dimensional sensor, and cannot detect overhangs or cliff ledges. The other implication is that as the range to an object increases, the density of the point range estimates across its surface can decrease dramatically. At a distance of even 5m, the readings are becoming very sparse, and large sections of the surface will not be observed at all, even over multiple time steps. Likewise, many smaller, distinctive variations in the environment which could be very informative are completely missed, and are not usable to estimate the robot's precise pose.

To create a better set of observations about the world, we would like to include sensor fusion with other types of sensors, such as sonar or infrared sensors. Also, we would like to consider a greater reliance on computer vision techniques, not just stereo, to provide a good density of sensory information. These inclusions would entail much more complex observation models, and possibly storing more information within the map. However, we expect this approach would result in much greater accuracy, and more efficient use of particles to localize the robot correctly, while building a more complete map.

12.1.1 Adapting Better Stereo Vision

In the three dimensional version of DP-SLAM that we have proposed, stereo vision is already used as the primary range sensor. However, stereo suffers from a number of complications for our application.

One of the most difficult aspects of using stereo vision for SLAM is the lack of an appropriate observation model. As we have noted before, the depth map returned by stereo necessarily has dependent noise. This will naturally create difficulties in modeling the sensor. Perhaps equally important is that the amount of noise for any given depth estimate can change, depending on the texture in the scene. Using stereo vision, it is much harder to determine the distance to a blank wall than a textured one. However, current methods for stereo vision do not include any indication of the amount of certainty in a given observation, or possible alternate depth maps that could result from the given image pair. Quantifying the uncertainty in the depth maps, or providing some means of sampling from possible depth maps would greatly affect the potential accuracy of SLAM methods which use stereo vision.

As was noted earlier, another limitation of using stereo cameras as the primary sensor is the assumption of perfect calibration. While there exist many methods for achieving good stereo calibration, most of the more accurate ones are time consuming, and require significant human intervention. Given that the algorithm's performance does not degrade gracefully as the calibration accuracy decreases, it would be useful to develop a better method for correcting the stereo calibration in the field. This could either be tracked as extra parameters in the SLAM formulation, or could be placed in an external loop, such as the EM framework used for odometry calibration.

12.2 Proposal Distributions

As with all sampling based method, DP-SLAM is heavily reliant on a good proposal distribution for fast, accurate results. Maintaining a joint distribution over maps and robot poses creates a high dimensional space within which to sample, and keeping this space as small as possible is crucial to real-time performance.

Autonomous learning of the robot's motion model is a great benefit to generating accurate proposal distributions. However, this is specifically a single set of parameters for the proposal distribution, based entirely on the odometry readings. The possibility still exists to allow the motion model parameters to change, even while the robot is exploring, to adapt to changes in terrain or degrading hardware on the robot.

Also, the inclusion of the sensor data itself could be used to aid the generation of the proposal distribution. Refining the motion based upon the range sensors has been used by other researchers with different SLAM methods [11, 30]. It seems likely that there exists a similar method which can be applied to DP-SLAM.

Another area of future research includes the development of alternate sensors to use as a motion estimate. Odometry is known to be a very poor estimate of the actual motion of a robot. Further investigation is needed into the suitability of adapting or designing other sensors to solve the problem. The preliminary work for 3-D DP-SLAM indicates that the use of visual keypoints, such as SIFT, has tremendous potential for giving accurate motion estimates with less noise.

12.2.1 Adaptive Particle Numbers

Beyond the question of how to generate proposal distributions is the question of how to use them. Many situations arise where the uncertainty in the system is significantly lower than normal. In these cases, it is natural to question whether the resources should be distributed in the same way. In fact, perhaps even fewer particles can be used at these

times, thus freeing the processor for other useful tasks for the robot. Alternatively, the extra resources could be used for processing more data, in order to give a more detailed map of the environment in that area, and perhaps further reducing the uncertainty at later time steps.

In the opposite direction, it would be useful to know how many particles are needed to estimate the robot's pose for a given proposal distribution, to better handle those situations where the uncertainty has grown larger than normal. This can reduce the amount of drift in the map, and greatly reduce the possibility of losing track of the robot's pose during unlikely events.

In general, determining an appropriate number of particles for a given proposal distribution could greatly improve the reliability of DP-SLAM under particularly difficult or uncertain circumstances. In a larger sense, using an appropriate amount of resources can also allow the robot in general to perform other tasks more efficiently, and allow DP-SLAM to be a more useful tool for a complete autonomous system.

12.3 Alternative Map Representations

DP-SLAM gains much of its success by maintaining a distributed particle map over a simple stochastic occupancy grid. The formulation of distributed particle maps allows DP-SLAM to maintain many different map hypotheses efficiently. However, there is nothing in this formulation which is necessarily restricting us to the simplest grid representation. A large number of alternate map representations are possible, any one of which could potentially benefit the accuracy or the efficiency of the DP-SLAM algorithm.

12.3.1 Quad Trees

One of the most popular ways of improving the efficiency of occupancy grids is the use of quad trees. This method seeks to improve both the running time of line traces, as well

reduce the space required to store the map, by combining adjacent grid squares if they are identical in their makeup. A tree structure is maintained, effectively representing the grid at a series of exponentially finer resolution. At each level in the tree, if each sub-component does not agree on the occupancy for the larger grid square, the quad tree breaks the grid square up into the four component sub-squares.

Adapting this idea to DP maps presents a number of challenges. The primary challenge is that the grid squares in a DP map are very rarely identical. Since each grid square potentially has a large set of different entries, it is unlikely that adjacent grid squares agree across all possible hypotheses. Even for a single particle, the stochastic nature of the occupancy grid means that adjacent grid squares often have differing amounts of evidence to support an observation. Even under the best of assumptions, it seems unlikely that quad trees could present a significant advantage in terms of space requirements. However, it is possible that some level of speed may be gained, if the quad tree is formulated correctly.

Instead, let us concentrate on increasing the speed of line traces through the map, which will improve running time of the localization stage, empirically the most time consuming portion of DP-SLAM. If the quad tree is built by ignoring the amount of supporting evidence, grid squares could be generalized over their density values, ρ. The only situations where this value could be expected to generalize with any regularity is in areas observed to be empty, or in areas where no observation has been made at all. Fortunately, these two types of grid squares dominate the bulk of the map. The most straightforward approach would involve maintaining an entire observation list at each level in the quad tree, and the decision as to whether to look at the next level would be made individually for each particle.

There remain a number of issues regarding precisely how to implement this basic concept, particularly concerning how to maintain the map cache effectively. However, it is possible that the time required to perform the necessary line traces through the map could

be dramatically reduced. It is also possible that it could be used to help with map updates, with some further research.

12.3.2 Variable Map Resolution

Another issue that needs to be addressed is the possibility of variable resolution occupancy grids. Most existing sensors degrade in resolution as the distance increases. This occurs not only from a potential increase in the noise from the sensor, but also from the expansion of the field of view. Since the data points from a single observation do not cover parallel lines, the distance between the data points grows as the distance from the sensor increases. This leads to a sparse coverage of the environment at the far end of the sensor range.

To deal with this problem appropriately, particularly with very high resolution maps or long range sensors, it would seem appropriate to be able to represent the map at several different levels of resolution. Areas first observed at a great distance would be entered into the map at the lowest resolution, and as the robot approached closer to that area, the data could be resolved into a higher resolution. How exactly to generalize data both to higher and lower resolutions, in a principled and efficient manner, is an open question which is worth some serious investigation. This would be particularly useful for outdoor environments with a three dimensional map, both in terms of increasing the accuracy of the map as well as possibly increasing the practical speed of the algorithm.

12.3.3 Soft Updates

In DP-SLAM there is a slight inconsistency in the way that the sensor is handled. For purposes of localization, the observations are treated as noisy, to create a a more realistic probability for each observation, given the robot's pose and map. However, when updating the map, the sensor is treated as a deterministic sensor, and the observation is added to the map as if it were completely accurate. This is not a complete contradiction, as the map can

be viewed as an accumulation of evidence, and thus a predictor of what future observations might return.

In order to build a better map, particularly with greater motion between observations or a finer grid resolution, it may be useful to treat the sensor as a noisy reading when updating the map as well. This would entail making a soft update to the map, perhaps by distributing the observation of stopping the sensor across multiple grid squares. The original work on occupancy grids [22] developed a very good early treatment of this idea for sonar sensors. This could also be extended to include some degree of hypothesized data association, so as to attempt to exactly pinpoint the true locations of objects, as compared to where we can expect them to be observed in future sensor readings.

12.3.4 Improved Priors

In earlier sections, we describe the treatment of previously unobserved grid squares, and the prior that is used for determining their occupancy during localization. The development of this prior has largely been intuitive and empirical. After using our knowledge of the laser range finder in general, we were able to make a reasonable assumption about the behavior of a sensor in the absence of previous knowledge, which we have found to works well in practice. However, significant improvement on this treatment of unknown space is likely possible.

More examination of the proper treatment of previously unobserved squares could potentially develop a superior method. Experiments to determine the likelihood of a grid square's occupancy, based on other nearby observations, might uncover a useful set of dependencies. This could lead to a treatment very similar to soft updates, or variable resolution. Another approach that has been recently suggested for occupancy grids is the use of polygonal random fields to provide a principled set of priors for the unobserved areas in the map [39]. Given the significant amount of new area which is observed at each time step,

it is could be very valuable to extract any possible extra information which could improve the accuracy of the map.

12.3.5 Spheres of Influence

Previously, all of the discussion of occupancy grids has, not surprisingly, assumed a discrete, grid-like structure to store the evidence in. However, there is nothing specifically holding us to this structure. One of the most interesting possible directions to investigate for mapping is the possibility of other representations of a metric map.

One possible extension to occupancy grid involves removing the inherent discretization of the world. While this is a very convenient form for storing the information collected by the sensors, it can lead to possible biases or loss of information across the discrete boundaries of the grid.

Consider a map representation where the information is still stored at regular intervals along a grid-like structure. However, instead of each point representing a finite grid square, over which this data is the only pertinent set of observations, we can allow the data points to have overlapping areas of influence. Thus the probability that a given location is occupied is a combination of several of the nearby data points, weighted by some function of the distance from the given location to the data point. This allows us to not only avoid the discretization problems inherent in an occupancy grid, but also suggests a partial solution to the treatment of unobserved areas. This formulation also provides a natural progression across multiple resolutions.

Several issues would still need to be investigated in this formulation of the map. Certain technical issues would need to be handled, such as the appropriate weighting function used to determine the influence of a particular data point. Also, a new method would have to be developed to determine how to compute the probability of a line trace, now that the density of an area is a smooth function, rather than a series of step functions. It may become

necessary to compute a full integral of the probability of stopping the sensor, rather than taking a sum of discrete steps. However, even an approximation of an integral could be more accurate than the implicit approximation introduced by the discrete occupancy grid.

This map representation, or others like it, is likely to require significant amounts of research to develop fully. However, the potential rewards for a more dynamic and more exact map representation could be very useful, not only for SLAM, but other robotic applications as well.

12.4 Active SLAM and Exploration

Typically, SLAM is treated purely as a passive algorithm; the robot's motion and sensors are left under the control of another process on the robot. This restriction is not necessary, however, and interesting control questions arise when the uncertainty of of SLAM is considered in determining the best actions for the robot.

In the field of pure localization, there is some research into the problem of *active localization* [40]. For this problem, the localization algorithm is recognized as having difficulty under certain circumstances. To reduce the probability of the robot becoming lost during a trajectory, the potential uncertainty in the robot's pose is taken as an additional cost while performing path planning for the robot.

Interesting decisions could result from attempting to solve the same problem for SLAM. While many of the same situations which are unfavorable for localization would also be undesirable for SLAM, there exist many new and more complicated problems for SLAM. The precise path used to arrive at a given location has a significantly greater impact on SLAM than on pure localization. This could possibly lead to better paths which include several small loops in the trajectory, since this could greatly reduce the uncertainty in the map, allowing the robot to arrive at the desired location with more certainty.

In addition to optimizing path planning with respect to the difficulties of SLAM, there

are other questions that concern integrating SLAM with control strategies. One area which has received very little rigorous attention is the problem of exploration. In many situations, such as scientific missions, search and rescue, or surveillance, the collection of new information is the very goal of the robotic motion. In these cases, we would like to determine the optimal path for the robot to take that provides the most information about the environment, and expands the map as much as possible. In the cases of exploration, the robot must prioritize which unknown areas to visit next, and how best to place itself so as to observe the most new information, while still maintaining a high degree of reliability and accuracy for the map and position of the robot.

12.5 Principled Loop Closing

A large number of existing SLAM methods have loop closing as an explicit feature of the algorithm. Whenever the robot revisits an area by a different path, there exists additional information which can be leveraged for improved accuracy. However, the majority of these methods perceive loop closing as a separate event from the rest of mapping, and incorporate the additional information in a weak and unprincipled manner.

DP-SLAM does not currently address the loop closing issue directly, as we have concentrated on improving the overall accuracy of the algorithm, and use loop closure as a an evaluation measure, rather than as a way to fix problems. In particular, the hierarchical mapping method leverages much of the same information used in explicit loop closing techniques, in a more principled approach. However, other approaches to the same problem could possibly result in better solutions.

One potential method to exploit the information in closing a loop is very similar to the hierarchical mapping method already described. However, instead of using DP-SLAM at all levels, it may be a better idea to only use the particle filter at the lowest level. At the higher level, we could instead use an extended Kalman filter to maintain the relations

between the various local maps. This would allow the high level to choose a variety of methods to "sense" the various local maps. Landmarks could be identified as features within the local maps, keeping in the spirit of most EKF methods. Alternatively, a restricted method of global localization could be performed on the different local maps, as it became likely that robot had returned to that area.

Bibliography

[1] A. Howard and N. Roy, "The robotics data set repository (radish)," 2003. [Online]. Available: http://radish.sourceforge.net/

[2] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[3] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun, "Sonar-based mapping with mobile robots using EM," in *Sixteenth International Conference on Machine Learning (ICML-99)*, 1999.

[4] S. Thrun, "A probabilistic online mapping algorithm for teams of mobile robots," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 335–363, 2001.

[5] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333–349, 1997.

[6] J. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, 2000.

[7] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli, "Local metrical and global topological maps in the hybrid spatial semantic hierarchy," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2004.

[8] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. Berlin: Springer-Verlag, 2001.

[9] S. Thrun, "Probabilistic algorithms in robotics," *AI Magazine*, vol. 21, no. 4, pp. 93–109, 2000.

[10] K. Murphy, "Bayesian map learning in dynamic environments," in *Advances in Neural Information Processing Systems 11*, 1999.

[11] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003, pp. 1151–1156.

[12] M. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acupulco, 2003.

[13] G. Welch and G. Bishop, "An introduction to the kalman filter," Department of COmputer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, Tech. Rep. TR95-041, 1995.

[14] P. Cheeseman, P. Smith, and M. Self, "Estimating uncertain spatial relationships in robotics," in *Autonomous Robot Vehicles*, 1990, pp. 167–193.

[15] Y. Liu and S. Thrun, "Results for outdoor-SLAM using sparse extended information filters," 2003.

[16] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Uncertainty in Artificial Intelligence (UAI)*, 2000.

[17] J. Nieto, T. Bailey, and E. Nebot, "Scan-SLAM: Combining EKF-SLAM and scan correlation," in *5th International Conference on Field Robotics (FSR)*, June 2005.

[18] S. Thrun and A. Bucken, "Integrating grid-based and topological maps for mobile robot navigation," in *Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996, pp. 944–950.

[19] S. Thrun, J.-S. Gutmann, D. Fox, W. Burgard, and B. Kuipers, "Integrating topological and metric maps for mobile robot navigation: A statistical approach," in *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998, pp. 989–995.

[20] D. Hähnel, D. Fox, W. Burgard, and S. Thrun, "A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[21] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *1985 IEEE International Conference on Robotics and Automation (ICRA-85)*, St. Louis, Missouri, Mar. 1985, pp. 116–121.

[22] M. C. Martin and H. Moravec, "Robot evidence grids," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-96-06, March 1996.

[23] H. Moravec, "Certainty grids for sensor fusion in mobile robots," in *Sensor Devices and Systems for Robitcs*, 1989, pp. 243–276.

[24] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research*, vol. 11, 1999.

[25] D. Fox, "Markov localization: A probabilistic framework for mobile robot localization and navigation," Germany, 1998.

[26] J. H. Leonard, , and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," in *IEEE Transactions on Robotics and Automation*, June 1991, pp. 376–382.

[27] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simulataneous localization and mapping without predetermined landmarks," in *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acupulco, 2003.

[28] C. Kollman, K. Baggerly, D. Cox, and R. Picard, "Adaptive importance sampling on discrete markov chains," Los Alamos National Laboratory, Los Alamos, NM, Tech. Rep. LA-UR-96-3998, 1996.

[29] D. Hähnel, W. Burgard, D. Fox, and S. Thrun, "A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[30] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 2443–2448.

[31] J. Borenstein and L. Feng, "Umbmark - a method for measuring, comparing, and correcting dead-reckoning errors in mobile robots," University of Michigan, Ann Arbor, MI, Tech. Rep. UM-MEAM-94-22, 1994.

[32] R. Voyles and P. Khosla, "Collabrative calibration," in *IEEE International Conference on Robotics and Automation (ICRA)*, Morgan Kaufmann, 1997.

[33] N. Roy and S. Thrun, "Online self-calibration for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, Morgan Kaufmann, 1999.

[34] W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence*, vol. 114, no. 1-2, pp. 3–55, 1999.

[35] N. de Freitas, R. Dearden, F. Hutter, R. Morales-Menendez, J. Mutch, and D. Poole,

"Diagnosis by a waiter and a Mars explorer," in *IEEE Special Issue on Sequential State Estimation*, 2003.

[36] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden markov model: Analysis and applications," *Machine Learning*, vol. 32, no. 1, pp. 41–62, 1998.

[37] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[38] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision ICCV*, 1999, pp. 1150–1157.

[39] M. Paskin, "Robotic mapping with polygonal random fields," in *Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI-05)*, Edinburgh, Scotland, 2005.

[40] D. Fox, W. Burgard, and S. Thrun, "Active markov localization for mobile robots," in *Robotics and Autonomous Systems*, vol. 25, 1998, pp. 195–207.

Biography

Austin Eliazar was born September 1^{st}, 1979 in Gainesville, FL. He received his Bachelor of Arts from New College of Florida in May, 2001.

His published papers include “DP-SLAM” (IJCAI-03), “DP-SLAM 2.0” (ICRA-04), “Learning Probablistic Motion Models for Mobile Robots” (ICML-04), and “Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps” (NIPS-05), all of which were co-authored with Ronald Parr.